MOSIS

-

Modelling Semantically Interoperable Web Service Compositions

Jon Myrseth

Master thesis submitted as part of the Cand. Scient degree in Informatics
at the Department of Informatics, University of Oslo.

February 2004

Preface

I would like to thank Arne-Jørgen Berre for all the help he has provided throughout the development of this thesis, his ideas and opinions have been invaluable. I would also like to thank Christoffer Daae-Qvale, Hans Stubberud and Hans-Gunnar Vold who participated in the development of the MODUSA case example. The cooperation turned out to be a stimulating environment were I could test and evolve my ideas. For useful comments in the last phase of this thesis I would like to thank David Skogan and Sturle Helland.

Last, but not least, I would like to thank Trine for all her love and support. It would not have been possible without you.

Executive summary

The use of information from different, often distributed, sources is part of many applications today, and in many cases these are heterogeneous sources developed independently without the intention of serving one specific application. This introduces the problem of semantic heterogeneity. Different sources are usually based on different vocabularies even in cases where they operate within the same domain. Data items defined in one source may have different definitions in another source although they really are the same (synonyms), and items that are different in reality may have similar definitions in two different sources (homonyms). There is a need for using these sources with regards to the meaning of the items in question not how they are defined in the sources, thus achieving what is called “semantic interoperability”.

The distributed sources that will be treated in this thesis are Web Services. Web Services provide a programmatic access point through the web to a set of related functionalities, and these can be combined to create a new Web Service with added value. The combination of Web Services is called a Web Service composition. Web Service composition impose several semantic interoperability issues. First of all Web Services need to be discovered, you can use it if you don't know it exists, and this requires a semantical match between the descriptions of a requested service and the descriptions of existing services. Secondly, when you compose service, you need to make sure that information from one service that will be used as input to another service is semantically related to the input.

In order to deal with semantic interoperability issues concerning Web Service compositions we propose MOSIS (**MO**delling **S**emantically **I**nteroperable web **S**ervice compositions). MOSIS makes use of UML models and principles from Model Driven Architecture (MDA) to model the discovery, the composition and the ontologies used to define semantically the concepts involved. As an effect of using UML modelling we claim that both the process of using ontologies and defining them will become easier for developers unfamiliar with ontology description and developing. MOSIS facilitates functionality based Web Service discovery which makes it possible to match descriptions of a Web Service with descriptions of existing Web Services. In addition to this MOSIS provides measures of dealing with mappings that have to be developed in order to make sure that output from one service can be used as input to another.

Table of Contents

Table of figures

Table of tables

Part I

-

Semantic interoperability and key technologies.

1. Introduction.

The use of information from different, often distributed, sources is part of many applications today, and in many cases these are heterogeneous sources developed independently without the intention of serving one specific application. This brings up the problem of semantic heterogeneity. Different sources are usually based on different vocabularies even in cases where they operate within the same domain. Data items defined in one source may have different definitions in another source although they really are the same (synonyms), and items that are different in reality may have similar definitions in two different sources (homonyms). There is a need for using these sources with regards to the meaning of the items in question not how they are defined in the sources, thus achieving what is called “semantic interoperability”.

1.1 The area of research

This thesis will focus on how to deal with semantic interoperability issues that arises when using distributed, heterogeneous sources. These issues arise from the fact that these sources are developed independently by different developers and it is not always obvious what the meaning of the elements involved are, making it difficult for a developer to make use of the source. We will in this thesis explore these issues and investigate how we can deal with them.

1.2 The research problem and why this is worthwhile studying

As mentioned above the research area for this thesis is distributed systems and semantic interoperability issues that arise when trying to make these systems work together. The specific type of distributed systems we will investigate are Web Services. Web Services is a technology that provides a programmatic interface to software systems that is independent of the underlying operating system and the programming languages involved.

This thesis focuses on how to achieve semantic interoperability when developing Web Service compositions. Web Service compositions are Web Services or other applications that make use of already existing Web Services composed together. Web Service compositions involve a range of semantic interoperability issues, resulting from the fact that these compositions will be created by different developers than those who created the Web Services in the composition.

Web Service composition promises to revolutionize the Internet as we know it today. Being able to pick services off the Internet that fulfills a specified task you need to perform in your own system in cooperation with each other enables a range of opportunities. Potentially Web Service compositions might shift the focus of developers from reinventing the wheel by writing code for every task to using the wheel over and over by orchestrating and composing Web Services fulfilling the tasks that need to be performed. However there is still a long way until we reach this potential situation, and going there means first of all untangling semantic interoperability issues.

1.3 The objective of the thesis; how far do we hope to advance the knowledge in the field.

In this thesis we present MOSIS (**MO**delling **S**emantically **I**nteroperable web **S**ervice compositions) our proposal to an approach dealing with Web Service compositions. MOSIS is the result of an evaluation of existing approaches to Web Service composition in terms of seven requirements we have defined for Web Service composition approaches. With MOSIS we have tried to improve the points where the existing approaches not fully fulfill the requirements.

MOSIS is an approach that can be undertaken using the development tools of today with relatively minor modifications. The founding block of MOSIS is the use of UML models to bring semantic descriptions up from a rather challenging form to a form that is recognizable by most developers. The use of UML in all the phases of a Web Service composition puts the focus on having the developer being able to interpret the semantics of a Web Service and not only on having machines interpret semantic descriptions.

1.4 Research method in brief

Central in this thesis is an example case that culminated in a client making use of four Web Services. This example case is used throughout the thesis, first in chapter 6 as basis for defining the requirements for an approach to Web Service composition. The issues we found concerning the construction of the client gave us many results in terms of what the real challenges are. These requirements where used to evaluate two of the dominating approaches for Web Service composition and thus identifying what these approaches lack in order to master Web Service compositions.

In chapter 8 the example case is used to illustrate our proposed approach to Web Service composition and showing how the different challenges are met by our proposal. By using an example we aim to bring MOSIS out from a purely theoretical world and show that it in fact can be used in the real world.

1.5 Structure of the thesis

The thesis is divided into three parts. Part I concerns the problem area, defining such things as semantic interoperability and Web Services, and covers the first five chapters. Chapter 2 defines the problem area, addressing interoperability in general and semantic interoperability specifically. Chapter 3 gives an overview over the most important approaches dealing with semantic interoperability issues and introduces a sort of common denominator, ontologies, that is the most promising technology resulting from the research in semantic interoperability. Chapter 4 provides an introduction to the technologies that will be used for the solution proposed in this thesis. It also look into the existing approaches for dealing with Web Service composition. Chapter 5 will introduce a case example used to work out the requirements and to illustrate how the proposed solution in chapter 7 can help developers dealing with Web Service compositions.

Part II covers chapter 6, 7 and 8, and deals with our proposal for a dealing with Web Service composition. Chapter 6 takes us toward our approach by identifying the requirements for such an approach and evaluating the existing approaches. Chapter 7 presents MOSIS, our proposed approach for dealing with Web Service composition. Chapter 8 illustrates MOSIS using the case example from chapter 5.

Part III covers the last two chapters and sums up this thesis. Chapter 9 provides a conclusion to the findings from part II and gives some pointers to further work in the Web Service composition field. Chapter 10 is a bibliography covering the resources this thesis is based on.

1.6 Summary

This chapter dealt with the topic and goals of this thesis. Semantic interoperability among Web Services cooperating in a Web Service composition will try to be achieved by applying MOSIS, our proposal for dealing with Web Service compositions, which makes use of models to increase the understanding of semantic descriptions among developers.

2. Interoperability

Getting independently developed systems to work together has provided developers with major challenges for a few decades now, and still is. The amount of effort going into these challenges has added to the computer industrys ever growing collection of "buzzwords" by introducing the umbrella term **interoperability**.

The term interoperability has grown to encompass a broad range of issues and potential issues that come into play when trying to get systems to work together. This has made interoperability a less comprehensive term than it could be. In order to get a full understanding of what interoperability is, it is necessary to stop treating interoperability as a label for all issues concerned with getting systems to work together in performing a task, but instead realize that interoperability is multifaceted and goes beyond merely the systems involved.

This chapter will try to sort out what interoperability is by looking at definitions of the term and explore the different levels and categories of interoperability before coming to the issue at hand semantic interoperability, which will be the main focus of this thesis. As we will see, when trying to get systems to work together in performing a task, semantic issues have their own way of fogging up the picture. Semantics are part of many of the different interoperability issues that will be explored in the next section.

2.1 Defining interoperability.

Interoperability is often treated monolithically or simply from a system perspective, but in order to focus attention on specific interoperability problems more factors need to be taken into consideration. Before giving a more detailed overview of the interoperability landscape, we'll take a look at the definitions that can be found by searching the Internet. Here are a few of them, illustrating the most common flavors to be found.

Interoperability definitions

- *"The ability of software and hardware on multiple machines from multiple vendors to communicate".* [FOLDOC]

- 1. "*The ability of systems, units, or forces to provide services to and accept services from other systems, units or forces and to use the services so exchanged to enable them to operate effectively together".*
  2. "*The condition achieved among communications-electronics systems or items of communications-electronics equipment when information or services can be exchanged directly and satisfactorily between them and/or their users. The degree of interoperability should be defined when referring to specific cases".* [ITS]

- *"Interoperability is the ability of two or more systems or components to exchange information and to use the information that has been exchanged".* [IEEE90]

While these definitions are not that dissimilar, they have subtle differences and all of them try to say it all in one sentence. Looking at these definitions put together, the problem is reduced to finding out what it means to provide, accept, use and exchange services and information, or as the first definition states, what it means to be able to communicate. Before going into this, we'll first try a different approach by looking at the definitions of the terms interoperability are constructed from.

Interoperability can be broken up into *inter* and *operability*, inter meaning, *between, among, in the midst of, within, mutual, mutually, reciprocal, reciprocally*, and operability meaning, *being such that use or operation is possible, possible to put into practice, practicable.* The verb operate has the meaning, *to perform a function, work, to exert an influence, to produce a desired or proper effect* [AHDIC]. Meanings related to medicine and the military have been left out here .

Putting these definitions together, interoperability or interoperate is about a relation between something; businesses, software enterprise systems, information systems in general, etc. The relation is about performing or producing something, being able to perform or produce something as a result of the relationship.

In summation it is safe to say that interoperability is a term covering many aspects, and because of this it is difficult to reduce the term into a simple definition. Non of the definitions above are precise

enough to fully grasp the meaning of interoperability. Taking a closer look at the implications of the definitions brings us one step closer to a more precise understanding. Interoperability does not only cover technological aspects, but also business and legal aspects.

"One can think of interoperability as a jigsaw puzzle comprised of several integral pieces, each of which may, in turn, be comprised of smaller pieces" [PKI00].

Another important aspect of interoperability is the fact that for systems to be truly interoperable, it means that they have to continue to be so over time. Changes in the context these systems are working under should preferably not interfere with the systems ability to continue to perform the task they were performing before the introduction of the changes.

2.1.1 Interoperability on different levels

Interoperability is quite often seen from a system perspective, and while this is important too, there are other considerations that has to be taken as well. This can be the business context the systems are working in, legal aspects and so on. In this section we will look at a coarse classification of interoperability issues that consists of three categories reflecting our view of the interoperability landscape. The business level, the application level and the communication level. These levels can be further divided, but for this discussion, these three levels suffice. Business level interoperability concerns such things as business model, business processes, trading partner agreements, and legal aspects. Application level interoperability concerns such things as application logics, data formats, control data, etc. Communication level interoperability concerns such things as the protocols followed by the interoperating systems, i.e. how to exchange data.

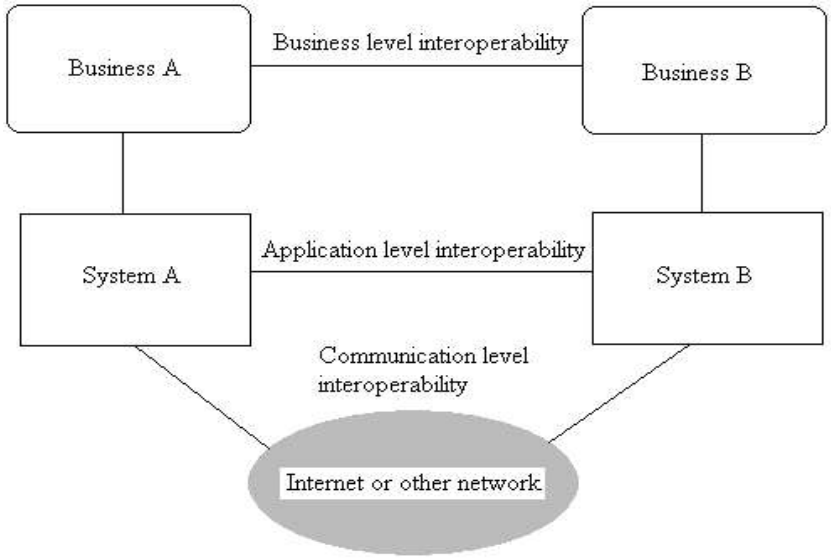

Figure 1 - Levels of interoperability

Figure 1 illustrates the three levels we'll focus on here. Traditionally most of the literature has been focused towards what is here called application level interoperability.

2.1.1.1 Interoperability on the business level.

With business level interoperability we mean all the things outside of the system itself that influences the system, and needs to be in place in order to achieve full interoperability among businesses. This includes the business models, business processes, trading partner agreements, legal aspects, etc. Some even include the user aspect, by taking into consideration the users assessment of interoperability, since it is ultimately the users who benefit when systems interoperate [WM01]. Nevertheless, regardless of how you categorize further, the important lesson is that outside factors influence the interoperation between systems.

Before two businesses[1] can start to make their respective software systems interoperate there needs to be a reason to do so. Both businesses have business models that they follow and the purpose of the interoperation will have to fit into both business models. Business processes are even more important as they define how the business conducts their tasks, and these need to be understood so that the interoperation fits nicely into the processes of both businesses. In addition, since business might cross both business domains and country borders, legal issues needs to be considered as well.

All the above mentioned aspects will in some way or another influence the software systems of the businesses and can therefore hamper interoperability efforts. The amount of effort that needs to go into making systems interoperable will also depend upon the "closeness" of the domains that try to achieve interoperability among their systems. Businesses operating in relatively close domains, will have a greater chance of achieving interoperability since the above mentioned aspects are more likely to be aligned than if the businesses are operating in domains that are more or less unrelated.

2.1.1.2 Interoperability on the application level.

Traditionally literature has treated interoperability from a system perspective, and that treatment falls under what we here call application level interoperability. Just as the business level, this level also has different aspects to it. Sheth distinguishes between four different classifications of interoperability problems that can occur during the interchange between information systems [SL03]:

- **System**, heterogeneous hardware and operating systems
- **Syntactic**, different representation languages and data formats
- **Structural**, heterogeneous model representations
- **Semantic**, different meaning of terms used in the interchange.

Differing operating systems and hardware is a situation that will affect most interoperating systems, however most of these issues can be dealt with by translations in much the same way as you would

1 Could be more than two, but for the example we limit us to two.

deal with translations between different representation languages and data formats. Conversion between A and B is something computers are good at, since it only involves mathematical calculations. System and syntactic issues are more easily dealt with than structural and semantic issues, which will be the topic for section 2.2.

2.1.1.3 Interoperability on the communication level.

Communication between computers is dealt with by the use of protocols such as TCP/IP, but this is not the only protocol suite out there. There are several protocols that might be used by the systems involved and some translation between the protocols is necessary in order to achieve interoperability on the communication level. These issues have been dealt with successfully in the past and remains as a non-trivial issue.

2.2 Defining semantic interoperability.

Semantic interoperability should be easier to define as it refers to a subset of what the general term interoperability encompasses. But is it so? As we will see semantic interoperability relates to several of the levels introduced earlier, and while the definition might be easier to get a grip of, it certainly does not make for an easier problem to work with.

Before we deal with the definition of semantic interoperability, we'll do the same trick of looking at the terms it consists of by themselves. Since we have already dealt with the definition of interoperability, we'll look at the term semantic here, what are the semantics of semantic? Semantic; *of or relating to meaning, especially meaning in language. Of, relating to, or according to the science of semantics* [AHDIC]. *Of or relating to the study of meaning and changes of meaning; "semantic analysis"* [WORDNET].

Semantic interoperability by using the definitions of semantic and interoperability presented here is something that is able to perform or produce something in cooperation with something else in such a way that the meaning of everything involved has a common understanding by the parties and thus the software systems involved in the task.

Practically this means that the information, the means to manipulate the information, why and how this is done must have clear semantics that are understood in the same way by the parties involved. Business and legal aspects also come in to play when trying to achieve semantic interoperability. The business processes and the considerations that has to be made regarding laws, which can differ from country to country, needs to be understood in the same way by the parties and software systems involved.

Semantic interoperability is not just another level together with the levels presented in section 2.1.1, semantic issues exist in all these levels. A common understanding is needed whether it's on the business level, the application level or the communication level.

2.2.1 Semantic issues concerning business level interoperability.

On the business level the business processes of the businesses that want to have interoperable systems will have to be in alignment with each other. All business applications realizes business processes or parts of them, and in order to be in alignment the individual business processes will have to be understood by the other parties involved. If for instance one of the applications needs to have task A before it can do task B, other applications might have to wait to provide information that is needed to complete task B until the first application has completed task A. Business processes define why things are done and when they are done, and thus the semantics of why and when needs to be known by the interoperating systems.

2.2.2 Semantic issues concerning application level interoperability.

On the application level the applications that are interoperating needs to have a common understanding of the information they exchange. This means that an application receiving information from another application needs to know what that information means or else it has no way of knowing what to do with the information. If the information an application will receive is fixed in the sense that the semantics of the information is known by the developers who are making the applications interoperable the problem is reduced to creating conversions in case there are any system, syntactical or structural differences. If, however, the semantics of the information is not known it will have to be determined before the applications can interoperate.

2.2.3 Semantic issues concerning communication level interoperability.

Communication level interoperability depends on standardized protocols that have well-defined semantics, so the semantic issues concerning this level can be regarded as more or less solved. Translations between protocols can be done by using the specifications of the protocols to see what the different parts it is made up of are.

2.3 Semantic interoperability issues

Semantic interoperability issues arise when you are using models of the real world that are developed by different groups of people. In one case different terms might be used to represent the same thing, making the terms synonyms, and in another case the same term might be used to represent different things, making the terms homonyms (fig. 2). These conflicts are whats called naming conflicts by Ceri and Widom, who identify four categories of semantic conflicts [SL03]:

- **Naming conflicts**
- **Domain conflicts**
- **Structural conflicts**
- **Metadata conflicts**

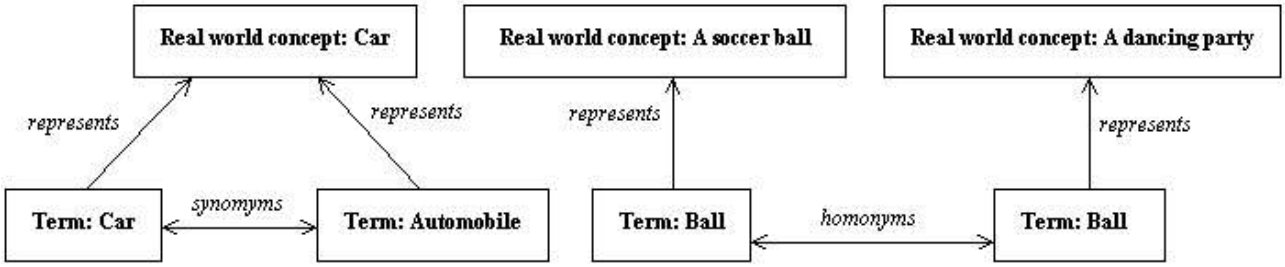

Figure 2 - Synonyms and homonyms

In between these two extremes there are other semantic relations that might exist between two terms used to represent concepts that are related. If one term represents a concepts that more general than a concept represented by another term, the first term is the other terms hypernym, and the latter term is the first terms hyponym (fig. 3).

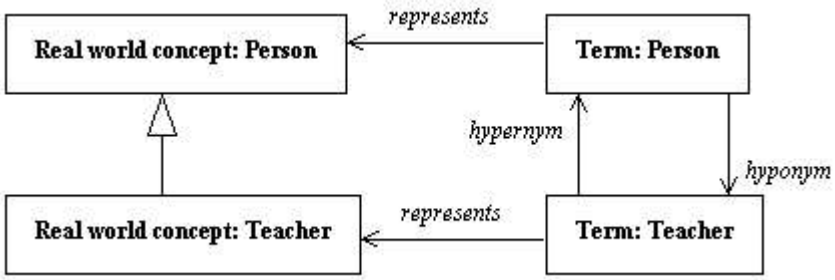

Figure 3 - Hypernyms and hyponyms

Domain conflicts are conflicts that occur when different reference systems are used to measure a value. For example, different currencies or different length units. Structural conflicts occur when different systems use different data organization to represent the same concept. For instance as the example in figure 4 shows, where the concept hand is modeled in two different ways.

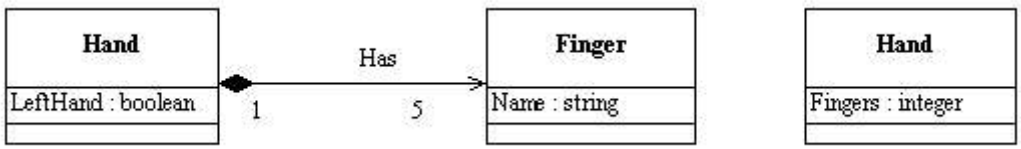

Figure 4 - Structural conflict

Metadata conflicts occur when concepts are represented as one type within the modelling type of one system and a different type within the other system (e.g. as at a schema level in one database and as an instance in another). Section 2.3.2 introduces a taxonomy of semantic similarity types to

put semantic conflicts in a context.

When the communication is between humans, we use a lot of context information to determine the semantics of what the other part is saying, the tone of voice and/or bodily expressions can alter the semantic of the term used. For instance, this is the reason for the use of smileys when humans are communicating textually through a computer, they can avoid misunderstandings since they can convey information about how a term is meant to be interpreted. When it comes to computer programs communicating some form of context information will also be necessary to determine the semantics of the terms used in the communication.

The challenge is to determine the relationship between the concepts, the terms used to represent these concepts are not important in themselves, but become important when trying to provide a mapping between terms that represent related concepts.

2.3.1 Semantic categories

In [SK92] Sheth and Kashyap introduce the concept of semantic proximity to characterize semantic similarity between objects. This qualitative measurement can be used to distinguish between different degrees of semantic similarity. The semantic proximity measurement is based on four concepts; context, abstraction, domain and state. The context is where semantic proximity holds, abstraction is the mapping between the objects domains and the state is the current value of the objects. The domains of the objects refer to the sets of values from which the objects can take their values. Given two objects O_1 and O_2, where object refers to a object in the model world which is a representation of the real world, the semantic proximity between them is defined by the 4-tuple given by;

$$\mathbf{semPro(O_1,O_2) = <Context, Abstraction, (D_1, D_2), (S_1, S_2)>}$$

where D_i is the domain of O_i and S_i is the state of O_i.

For the context(s) and abstraction(s) there are defined situations that affect the degree of semantic similarity between object. For context(s) the following situations apply;

- **All**, i.e. the semPro of the objects is being defined with respect to all possible contexts. The specific context does not need to be named.
- **Same**, i.e. the semPro of the objects is being defined with respect to the same context. The context must be explicitly specified in an instance of a semPro.
- **Some**, i.e. the semPro of the objects are being defined with respect to more than one context. The applicable contexts must be individually or collectively specified in an instance of a semPro.
- **Sub-contexts**, when the semPro can be defined in a previously defined context that is further constrained. The subcontext must be specified in an instance of a semPro.

- **None**, i.e. the objects under consideration do not exhibit any useful semantic similarity under any context.

For abstraction(s) the following situations apply;

- A **total one-one value mapping** between the domains of the objects, i.e. for every value in the domain of one object, there exists a value in the domain of the other object and vice versa. Also there is a one to one correspondence between the values of the two domains.

- A **partial many-one mapping** between the domains of the objects. In this case some values in either domain might remain unmapped, or a value in one domain might be associated with many values in another domain.

- A **generalization** mapping between the domains, i.e. one domain is more general than the other or both domains can be generalized/specialized to a third domain. There exists a total many-one mapping between the domains.

- An **aggregation** relation between the domains, i.e. one domain is part of the other domain. This can be expressed as a partial 1-1 mapping between the cross-product of the domains of the objects being aggregated and the domain of the aggregated object.

- **Functional dependencies**. They can be expressed as a partial many-one mapping between the cross-product of the domains of the determining objects and the cross-product of the domains of the determined objects.

- **Any**. This is used to denote that any abstraction such as the ones defined above may be used to define a mapping between two objects.

- **None**. This is used to denote that there is no mapping defined between two semantically related objects.

- **Neg**. This is used to denote that there is no mapping possible between two semantically unrelated objects.

The semantic proximity measurement can be used to define the following types of semantic similarity between objects;

- **Semantic equivalence**

  Two representations are referring to the same thing. This can be the same real world concept, the same attribute used to describe something, the same relation between real world concepts or the same operation. This means that in all contexts there must be a one-one and total mapping between the respective domains.

  semPro(O_1,O_2) = <ALL, total 1-1 value mapping, (D_1, D_2), _>

  Semantic equivalence is the strongest type of semantic similarity.

- **Semantic relationship**

  If, in all contexts, there can be established either a partial many-one mapping, a total many-one mapping or a partial one-one mapping between the respective domains, there is a semantic relationship.

  semPro(O_1,O_2) = <ALL, M, (D_1, D_2), _>

  where M = partial many-one value mapping, generalization or aggregation. Semantic relationship is weaker than semantic equivalence.

- **Semantic relevance**

  In both semantic equivalence and semantic relationship the mapping must be established in all contexts. It is not always possible to accomplish this, but if it's possible in the same context, we have semantic relevance.

  semPro(O_1,O_2) = <SAME, ANY, (D_1, D_2), _>

  Semantic relevance is weaker than semantic relationship.

- **Semantic resemblance**

  This is the weakest form of semantic similarity. When there are no mappings, but the respective roles are the same, we have semantic resemblance. A role is an aspect of context that is best explained by an example; if two almost equal classes are defined independently and both have the same attribute, but the attributes have incompatible constraints, these two attributes play the same role. [K96] defines semantic resemblance as;

  semPro(O_1,O_2) = <Context, NONE, (D_1, D_2), _>

  where Context = $context(O_1) \cup context(O_2)$
  and $D_1 \neq D_2$
  and role_of(O_1, Context) = role_of(O_2, Context),
  where role_of : $object \times context \rightarrow rolename$

- **Semantic incompatibility**

  This category is used when there are no contexts and no abstractions in which the domains can be related. The respective roles also have to be different.

  semPro(O_1,O_2) = <NONE, NEG, (D_1, D_2), _>

  where context = $context(O_1) \cup context(O_2)$ is undefined,
  and abstraction = NEG, signifying dissimilarity
  and D_1 may or may not be equal to D_2
  and role_of(O_1, context) and role_of(O_2, context) are incomparable.

2.3.2 Dealing with semantic interoperability issues.

When trying to deal with semantic interoperability issues, the procedure is to determine what terms have a represents related concepts, and then provide a mapping between the terms. If the terms involved are synonyms the mapping can be straightforward, but in many cases additional computations will have to be made. For instance, in one system a concept might be called “Fee” and have the currency type “US Dollars”, and in another system the same concept might be called “Cost” and expects a “Euro” representation. In this case, the semantic challenge is to determine that “Fee” and “Cost” are the same, and then the mapping will involve a conversion between “US Dollars” and “Euro”.

The two possible arrangement of the mappings between systems is to either provide one mapping between each of the systems that need it, star arrangement, or to create one common representation that all systems will have to have a mapping to, wrapper arrangement. What arrangement to choose depends on the purpose of the collaboration among the systems, and the amount of change the collaboration might have to tolerate.

Figure 5 shows a star arrangement where all systems have a mapping to the other systems. It is not necessary to have a mapping to every system, but only the ones that need to interoperate. The main problem with this arrangement is change, imagine that you have to replace system A in figure 5, the replacement system then needs mappings to the systems it interoperates with, that means in this example three new mappings.

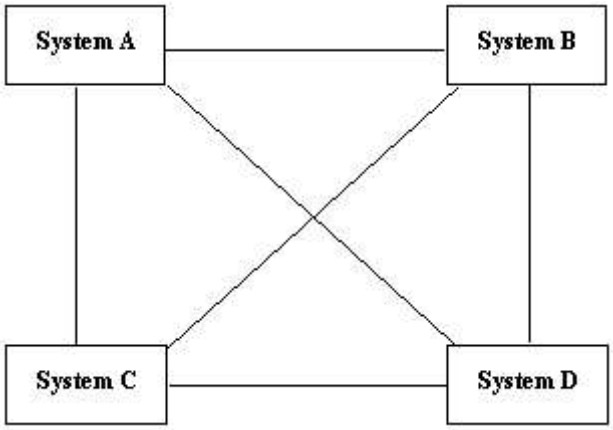

Figure 5 - Star arrangement

Figure 6 shows a wrapper arrangement where the systems wrap their internal format into a common representation format. This means that every system in the interoperating group of systems only needs one mapping, which consequently leads to only one new mapping in case of a replacement. In addition, if one of the systems for some reason stops working, finding out what data or processing that is missing is easier since you only have to check with the common representation format, and not all interoperating systems as you would in the star arrangement. The drawback with the wrapper arrangement is that the common representation format needs careful planning to deal with situations where a replacement or new system in the interoperating group needs data that is not found in the common representation format.

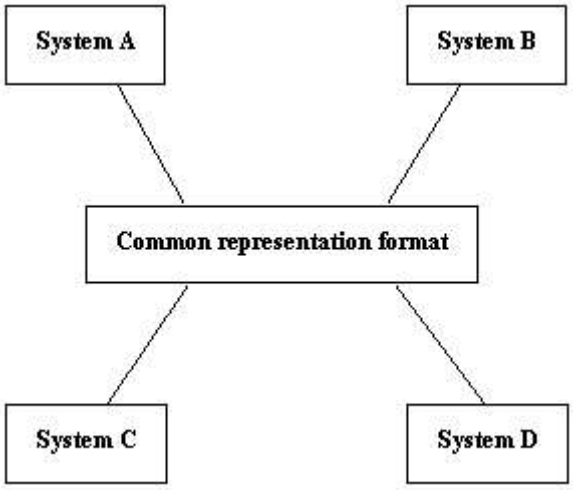

Figure 6 - Wrapper arrangement

2.4 Summary

This chapter introduced the interoperability term and gave a definition that tried to bring interoperability out of a buzzword stage where the term engulfs too much and split it up into three coarse levels that more precisely defines what interoperability is all about. We then went on to define semantic interoperability which is the main focus of this thesis. Semantic interoperability was also shown on the three levels we defined and we introduced the semantic categories that are defined by the SemPro function. Finally we looked at some ways of dealing with semantic interoperability issues by providing mappings between interoperating systems.

3. Approaches dealing with semantic interoperability issues

Semantic interoperability issues have been around ever since someone tried to get software written by different developers to work together. Many of these issues have been solved by developing standards for how to interact, where adhering to the standard guarantees you that your software will work with other software that follows the standard. Communication protocols, like TCP/IP are a good example of this, likewise are different interchange formats developed by business partners. The problem is that these standards work only for their specific tasks, and making standards for every possible task is, if not an impossible task, then at least a task that will take tremendous amounts of time and effort. The solution to this problem has been to standardize what you can and then find some method of translating between things that can't be standardized.

This chapter will look at some of the techniques that have been applied to deal with semantic interoperability issues. We have identified two main categories of approaches that broadly covers the approaches that exist. The first category covers approaches with a specific purpose and the second approaches with a general purpose. In the first category, specific purpose systems, a lot of the research on semantic issues comes from the database world, where trying to integrate different databases has been a major challenge for the last couple of decades. The techniques from this research does not only apply to database integration, but also to semantic interoperability issues in general. In the second category, general purpose systems, techniques from research on autonomous agents will be presented. The approaches presented in this chapter are not intended to be

representative of all approaches that exist, but aims to give an overview of the main flavors that has dominated the research on semantic interoperability. The conclusion of this chapter will introduce ontologies, the latest and most promising approach which can be said to dominate the research of today, and an approach that is useful for both of the categories.

3.1 Different approaches from the world of database integration.

This section will look at some of the approaches for dealing with semantic issues in the domain of database integration. Integration of different databases into one database has had the attention of researchers for a long time now, and many interesting results have come out of it that not only apply to the database world, but to anyone dealing with semantic integration or semantic interoperability issues.

When integrating data from different databases, there is a choice between constructing a new database with data from all the databases you want to integrate (data migration) or you can leave the databases as they are and create a system that retrieves the information you want from the different databases and then integrate the results. The main arguments for the latter option is that you don't have to change all of the applications using the databases you want to integrate. In a federated database system [SL90] all the databases are still autonomous and no changes to existing software should be needed. The federated system will lie on top of the databases and be able to access the information in all the sources with the appearance of being just one database system.

Regardless of how you integrate the databases one of the main issues to be resolved is the semantic issue. What is the meaning of the data? Probably the safest way to resolve these issues will be to get hold of all the developers of the information sources and get a full overview of the semantics. Unfortunately this is in most cases, if not all, impossible. The next step is to have someone manually asses all the information or at least key aspects to resolve the semantics. The main problem here is, as it also is in the first case, that the amount of information will in most cases be by far to large to overcome and the best solution will be some semi-automatic method, at least while we lack a perfect automatic solution.

Schema integration is a term used for integration within the database world. As the word indicates, the goal is to integrate all the schemas of the participating databases into one schema. This integrated schema will then be used to search the collection of databases by presenting a single interface to the user. One classical article on schema integration is Sheth and Larsons article on federated systems [SL90], which gives a good insight into how schemas can be integrated. They also mention the semantic issues that might arise, and claim that a schema integration process never can be fully automatic because of them. However, approaches have been used to deal with the semantic issues and some of them are presented below.

There are two different categories of approaches to schema integration, data transformation and query transformation. The choice between these alternatives is affected by several factors, including updatability and query performance. In either case, the conceptual differences between source and target schemas must be eliminated. This means that when databases are integrated it should be invisible to the user, for him or her it should appear to be only one system.

- **Data transformation**

  Data transformation means that the schemas of the participating databases are integrated into one schema, and any queries will work against this schema.

- **Query transformation**

  Query transformation means that the queries are split up into sub queries and distributed to the sources that are capable of answering the different sub queries. Results are then combined and returned to the user.

Whether it is data transformation or query transformation, the issues for how this should be done are much the same. Some of the typical approaches for resolving semantic issues are; procedural integration, relational algebra, and relational logic.

3.1.1 Procedural integration.

The procedural approach involves writing a piece of software to compute target relations from source relations. This is often done by writing so-called wrappers which hide all implementational details by extracting the information that's needed and presenting it to the overlying system in a uniform manner so that it can be integrated.

One consequence of wrappers is that all the information sources have to have a wrapper to be part of the system, and this means that information sources cannot be included in the system at run-time. How much time and effort that has to go into the development of a wrapper is dependent on many things, but it will take at least a few days [RS97].

3.1.2 Relational algebra and relational logic approaches.

In the *relational algebra* approach to semantic integration, target relations are defined as "views" of source relations in a language based on relational algebra, the most popular of these languages being Structured Query Language (SQL).

Advantages of this approach is the declarativity of the relational algebra languages and the fact that it brings the issues of integration down to the skill set of database administrators instead of programmers. However there are significant problems to this approach. Relational algebra languages have a limited expressiveness making some issues impossible to do something with.

As an example of this, consider the case of two relations providing information about personnel. Each has three attributes. The first relates social security number, department, and phone; the second relates social security number, department, and salary. Neither of these relations can be defined in terms of the other. The first cannot be defined in terms of the second because the second does not contain phone information. The second cannot be defined in terms of the first because the

first does not contain salary information. However, both contain department information; and, for the purposes of integration, it would be desirable to capture this relationship.

This problem can be solved by the relational logic approach. Like relational algebra languages relational logic languages are declarative and designed for schema mapping, but in addition to this they are more expressive and can capture relationships like the one in the preceding example.

Another problem with both the relational algebra approach and the relational logic approach is the connection to the relational data model, which make them unsuitable for any other data models.

3.2 The agent approach.

Agent is a term that originates from artificial intelligence research and has become widely popular throughout the computer industry recently. Research in agent-based technology has been conducted in many areas including; *"robotic agents, agents for automatic diagnosis, agents for control systems, agents for Internet-based information retrieval, agents for management systems, agents for telecommunication, agents for automatic negotiation, etc"*. [AL01]

This section will give a short introduction to what agent technology is, and look into some of the research that has been done on agent technology when it comes to semantic interoperability.

3.2.1 Agent technology.

Due to the increasing popularity of agent technology, the term agent is just like interoperability becoming a label which encompasses many things and therefore suffer from the loss of a definition that works for everybody. One of the most widely accepted definitions is given by Wooldridge and Jennings; *"an agent is a self-contained problem solving system capable of autonomous reactive, pro-active, social behavior."* [AL01]

In this definition lies the reason why agents are used to solve semantic interoperability issues. An agent will try to tackle semantic issues in the same manner as it tackles all other issues it encounters, by using problem solving techniques. The agent will of course need some form of rules along with metadata to solve its problems, it cannot just encounter the dataset, A, B, C and from this induce that A and B are the parents of C, it will need metadata and rules to induce that fact.

3.2.2 Agent technology and semantic interoperability.

This section will look at two approaches to achieving semantic interoperability form the world of agents. The first one is an approach called meaning negotiation, and the second one are FIPA compliant multi-agents systems (MAS).

3.2.2.1 The meaning negotiation approach

Meaning negotiation (MN) is based on the idea that agents interacting with each other use some method of negotiating towards a common understanding of the semantic of the concepts involved in the interaction. *"The idea of MN is that any real-world approach to semantic interoperability between autonomous entities (namely entities that cannot asses semantic problems by "looking into each other's head", like humans or software agents) should involve (among other things) a social process of negotiating an agreement on the content (semantics) and the speaker's intention (pragmatics) of a communication"* [AAAI02]. When any concept is unclear while trying to communicate something, the idea is that you can negotiate an agreement on what is the semantics of the concept. E.g. if you want information about hotels in Venice from a travel agent, you need to establish which Venice you want information about (Venice in Florida or Italy). Other things that might need to be determined are purpose of trip, the dates, price range, etc. After all these things are established appropriate information might be returned.

For instance, in an approach like meaning negotiation it is imaginable that one might use ontologies (section 3.4), in some way or another, to aid the negotiation process. Although in the description of the AAAI workshop on Meaning Negotiation it is presented as a divergent alternative to ontologies as they claim that a shared ontology *"... which - being a linguistic structure - is by definition compatible with many different interpretations*" [AAAI02].

3.2.2.2 FIPA compliant multi-agent systems

Multi-agent systems is another term with several different definitions. From the distributed artificial intelligent perspective, multi-agent systems are "*loosely coupled network of problem-solver entities that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity*" [MAS]. The interaction between the agents in a MAS are based on three key elements, a common agent communication language and protocol, a common format for the content of communication and a shared ontology (section 3.4).

The Foundation for Intelligent Physical Agents (FIPA) is a non-profit organization aimed at producing standards for the interoperation of heterogeneous software agents. FIPA have made available a series of specifications to direct the development of multi-agent systems. FIPA's approach to MAS development is based on a "*minimal framework for the management of agents in an open environment*" [MAS]. Multi-agent systems that adhere to the FIPA standards are called FIPA compliant multi-agent systems.

3.3 Similarities and differences of the approaches.

In addition to the apparent differences of the approaches presented above, the two main categories, database approaches and agent approaches, represent the separation of systems that are designed to perform a specific task and systems that are able to use problem solving methods to perform many different tasks.

All of the approaches above require some sort of minimal agreement on the semantics of the concepts involved. The next section will explore in more detail one popular technology for representing such a minimal agreement that can be used by all of the approaches presented so far.

3.4 Ontologies.

Ontologies come from the world of artificial intelligence and has attracted a lot of attention lately. They are being applied to wide range of areas, from e-commerce to integration. Although there are different ways of achieving semantic interoperability without depending on ontologies, they "*provide a shared and common understanding of a domain that can be communicated between people and application systems*" ([F01], p. 11), and has therefore become a popular research topic in various communities. As we'll see after defining what ontologies are, ontologies can aid both integration and interoperability efforts.

3.4.1 Definition of an ontology.

An ontology *"... is a formal, explicit specification of a shared conceptualization"*, as defined by T. R. Gruber ([F01], p. 11). This means that ontologies are structures capturing semantics and by using them semantic interoperability can be achieved.

3.4.2 The role of ontologies for semantic interoperability issues.

"The task of integrating heterogeneous information sources put ontologies in context. They cannot be perceived as standalone models of the world. They should rather be seen as the glue that puts together information of various kinds. Consequently, the relation of an ontology to its environment plays an essential role in information integration." [WVV01]

With respect to integration of data sources and interoperating systems that exchange data, ontologies can be used for the identification and association of semantically related information concepts.

Providing a link to an ontology can greatly improve any integration or interoperability efforts, however the perhaps greatest strength of ontologies is the possibility to add semantics to new developments such that any efforts in the future whether its integration or something else will be greatly simplified.

3.4.3 Different types of ontologies.

There are several types of ontologies, where the perhaps three most important ones are information source specific ontologies, capturing the knowledge most essential to one information source,

domain specific ontologies, capturing the knowledge most essential to the specific domain, and upper level or generic ontologies, capturing more general knowledge in terms of basic concepts and relationships [MD02]. Domain specific ontologies and information source specific ontologies are the easiest ones to develop. The greatest challenge is given by generic ontologies.

3.4.4 Using ontologies

In [WVV01] the authors distinguish between three different directions concerning how ontologies are used today. The options are; single ontology approaches, multiple ontologies approaches and hybrid approaches. Figure 7 shows the three options.

- **Single ontology approaches.**

  One global ontology provide a shared vocabulary for the specification of the semantics. Disadvantages with this approach is that all information sources need to have nearly the same view on a domain. If one source has another view, i.e. by providing another level of granularity, finding the minimal ontology commitment becomes a difficult task. Also changes in the information sources can affect the conceptualization of the domain represented in the ontology.

- **Multiple ontologies.**

  The development of multiple ontologies approaches was triggered by the disadvantages of single ontology approaches. In multiple ontology approaches each information source is described by its own ontology. Disadvantages of this approach is the lack of a common vocabulary which makes it difficult to compare different source ontologies. To overcome this an additional representation formalism defining the inter-ontology mapping is needed.

- **Hybrid approaches.**

  Hybrid approaches were developed to overcome the drawbacks of the single or multiple approaches. Each information source is described by its own ontology and in order to make the local ontologies comparable they are built from a global shared vocabulary. The shared vocabulary contains basic terms of a domain which are combined in the local ontologies in order to describe more complex semantics. Sometimes the shared vocabulary is also an ontology.

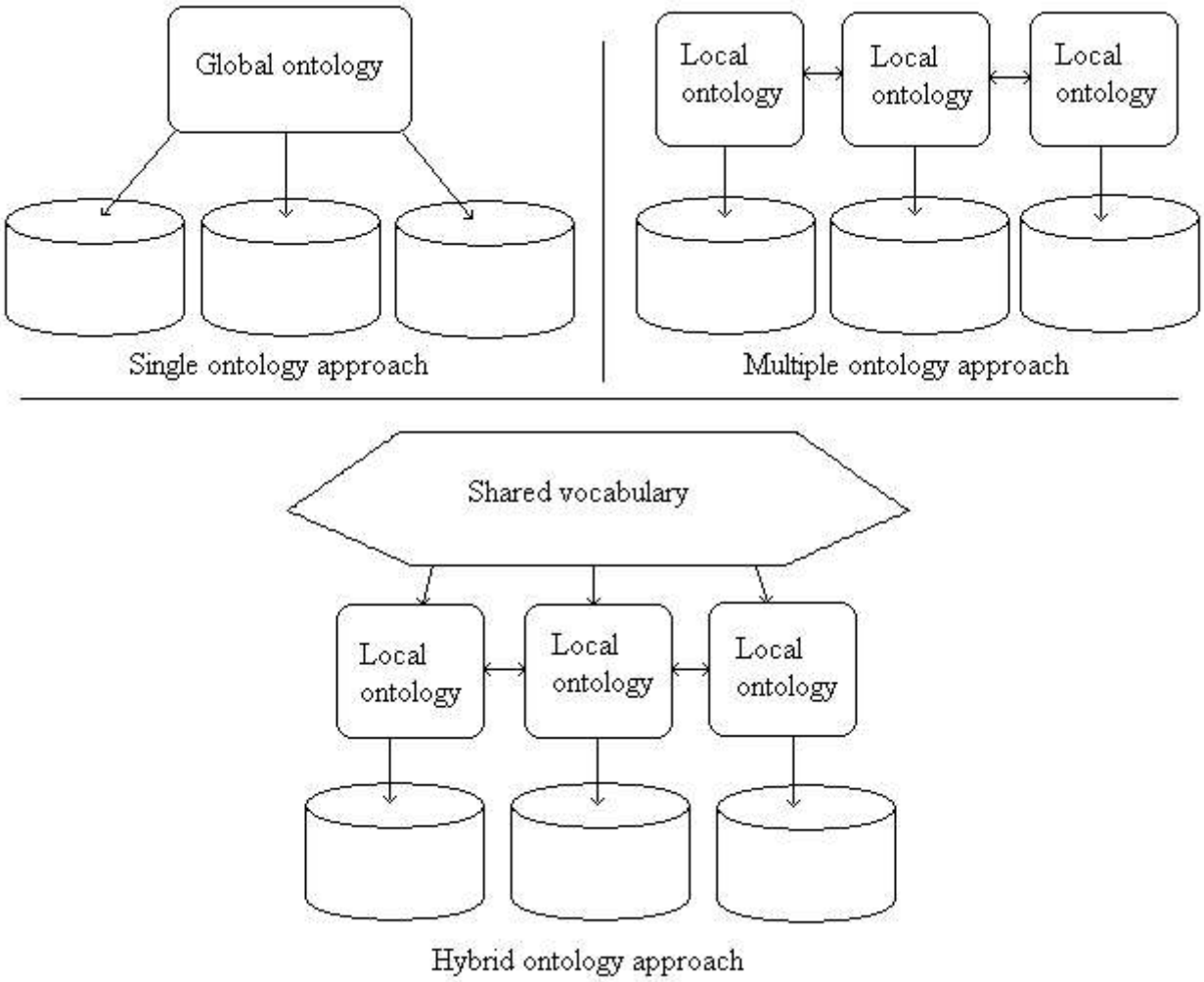

Figure 7 - Three possible ways for using ontologies for content explication .[WVV01]

The single ontology approach constitutes an impossible task as it means that all concepts must be defined in one single ontology for all information sources to make use of it. The multiple ontology approach pushes the problem from mapping between the data sources to the problem of mapping between ontologies. The hybrid ontology approach is the most promising of the three, but you still face the problem of what to put in the shared vocabulary, and how to define the concepts described there.

3.4.5 Building and evolving ontologies.

Building and evolving ontologies is facilitated by ontology representation languages and tools to support the management of ontologies and merging with other ontologies. Ontology representation languages have their roots in artificial intelligence research on techniques to describe the world. This research include formal mathematical theories such as, logic, algebra and graph theory.

The latest additions to the family of ontology representation languages are XML based languages, such as RDF(S), XOL, OIL, DAML, DAML+OIL, OWL and OPAL. Out of these, the ones that have gotten most attention lately is DAML+OIL and OWL, which are products of the "Semantic Web" community (Section 3.5), and therefore we will focus the attention towards these. These languages build on two techniques from AI research, frames and description logics. Frames are described by Minsky in this way; "*When one encounters a new situation (or makes a substantial change in one's view of a problem) one selects from memory a structure called a 'frame'. This is a remembered framework to be adapted to fit reality by changing details as necessary (Minsky 1975)*". ([L02], p. 214). Description logic is as the term implies a way of describing using logic, which makes it possible to perform automated reasoning.

- **OIL**

  OIL stands for Ontology Inference Layer and was developed in the On-to-Knowledge project. OIL supports logical reasoning with and about ontologies. OIL combines the widely used modelling primitives from frame-based languages with the formal semantics and reasoning services provided by description logics.

- **DAML**

  The DARPA Agent Markup Language (DAML) was developed in parallel to OIL. Its goal was to provide a common basis for the DAML program and is less influenced by formal logic than OIL. It mainly borrows from frame-based knowledge representation systems.

- **DAML+OIL**

  DAML+OIL is an result of an initiative to create a joint language on the basis of the then existing languages DAML and OIL.

- **OWL**

  OWL is the latest ontology initiative under W3C, and is based on the experiences made with DAML+OIL. OWL is built on top of RDF, and is used to semantically annotate web content, both pages and services.

Following the evolutionary path of the ontology representation languages, OWL is the natural choice for anyone building ontologies. The main reasons for this comes from OWLs connection with the Semantic Web vision, and the not so easily overlooked fact that OWL is becoming a W3C standard. However, the problem with OWL is its lack of maturity, only a few tools that support OWL have emerged, but when OWL becomes a W3C standard its likely that more and more tools will emerge.

Tools for building ontologies are not enough, they need to have management functionality or be complemented by ontology management tools. Management tools need to manage collections of ontologies, map and link between them, compare them, reconcile and validate them, merge them, and convert them into other forms. "*Ontologies may be derived from or transformed into forms*

such as W3C XML Schemas, database schemas, and UML to achieve integration with associated enterprise applications" [MD02]. The ability to convert ontologies into other forms is very important as they are mainly intended for machine-processing and not for human interpretation. If ontologies are to become widespread, they need to be developed by domain experts and not ontology representation language experts.

3.4.6 Standardization of ontologies.

Standardization plays a major role when it comes to ontologies. Trying to capture all knowledge in one super-ontology is not a feasible task, but standardization in different domains and committing to those ontologies as well as common ontologies published on the web ready to be used will be a further step towards semantic interoperability. Standardization of ontology languages is also important to avoid syntactical differences and conversion problems, in addition to conceptual differences.

Ontologies may aid in the process of achieving semantic interoperability, however being what they are they need commitment within the domain they are used to achieve their greatest potential. This may not be a feasible task as agreement on something when involving many interests usually brings more confusion on the table in terms of differing solutions. This is where standardization comes in. Although a slow process, given some time satisfactory standards might evolve.

3.5 The Semantic Web

One of the most influential efforts in trying to deal with semantic interoperability issues is known as the “Semantic Web” vision. The “Semantic Web” is an initiative by the World Wide Web Consortium (W3C) that currently are defining and have defined standards that will move the web as we know it today from a web of static pages to a dynamic web where content is understandable for both machines and humans.

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." [LHL01]

The idea behind the “Semantic Web” is make the content of the web meaningful to computers. Today the content of the web is designed to be readable by humans, but moving towards a web that computer programs can manipulate meaningfully will “*unleash a revolution of new possibilities*” as the authors of [LHL01] claim. Among the new possibilities envision by the authors is semantic web agents able to find providers of physical therapy sessions that are covered by insurance, rated as excellent or very good and within reasonable distance of the patient requiring such treatment. Enabling such tasks to be done by semantic web agents requires that information is found in a form that is interpretable by the software. The World Wide Web Consortium (W3C) are the ones developing standards to support the semantic web, and their definition on the semantic web is;

"The Semantic Web is the representation of data on the World Wide Web. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming". [W3CSW]

The ongoing work towards the vision of the "Semantic Web" has resulted in two technologies standardized by the W3C and progress towards the last piece of the puzzle. Extensible markup language (XML), adding structure, and the resource description framework (RDF), adding meaning, are the two technologies that exist and the last piece of the puzzle is ontologies which the web ontology language (OWL) is expected to contribute with.

- **XML**

"XML is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere." [W3CXML]

XML is a language that provides structure to data by using tags. Descriptions of the data in tags is defined in Document Type Definitions (DTDs) or in XML Schemas. Since XML is extensible there is not a fixed set of elements, you have to create your own tags or use tags defined by others. The web today is mostly made up of unstructured data marked up using HTML, and by using XML it is possible to add structure to the data. The following example is adapted from [O00].

```
Example:

HTML
<p>example.html is authored by John Doe</p>

XML
<document>
    <documentName>example.html</documentName>
    <author>
        <firstname>John</firstname>
        <lastname>Doe</lastname>
    </author>
</document>
```

As the example illustrates the XML version provides more structure, and the tags used can be described in a DTD or an XML schema to define such things that the element author is made up of the elements firstname and lastname.

- **RDF**

"RDF is a framework for metadata; it provides interoperability between applications that exchange machine- understandable information on the Web. RDF emphasizes facilities to enable automated processing of Web resources and as such provides the basic building blocks for supporting the Semantic Web." [W3CRDF]

While XML provides structure to the data, it says nothing about what the meaning of the structure is. Meaning is expressed by RDF by providing metadata. RDF metadata consists of statements corresponding to subject, verb and object of an elementary sentence. For example, the sentence, "example.html is authored by John Doe", can be broken up into; subject: *example.html*, predicative: *is authored by* and object: *John Doe*. In RDF resources are represented by Uniform Resource Identifiers (URIs), and all subjects must be resources. Objects on the other hand can be either resources or literals. The string "John Doe" is an example of a literal. Using URIs for the resources example.html and John Doe gives the following RDF expression for the sentence "example.html is authored by John Doe".

```
Example:

RDF
<document>
    <rdf:Description about="http://www.example.no/example.html">
        <documentname>example.html</documentname>
        <author>
            <rdf:Description about="http://www.example.no/johndoe">
                <firstname>John</firstname>
                <lastname>Doe</lastname>
            </rdf:Description>
        </author>
    </rdf:Description>
</document>
```

The structure of the statements in RDF turns out to be a natural way to describe the vast majority of the data processed by machines [LHL01].

- **OWL**

"OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web." [W3COWL]

RDF adds semantics to structured web documents, but this is not enough to identify semantically related concepts described by different web documents. The third basic component of the semantic web, ontologies, aim to provide a solution to this. The W3C is currently in the process of preparing their web ontology language (OWL) for W3C recommendation [W3COWL].

3.6 Summary

This chapter has given some insight into the ways of dealing with semantic interoperability issues, specifically from the database community and from the agent community. We concluded with the fact that most approaches for dealing with semantic interoperability issues require some form of common reference system, where upon we introduced ontologies, which is currently the most promising way of creating a common reference systems. In the end of the chapter we introduced the "Semantic Web" vision which is very influential in the work towards semantic interoperability.

4. Web Services and MDA

This chapter introduces Web Services and Model Driven Architecture (MDA). Web Services is the technology that provides the distributed setting in which we want to propose an approach to deal with semantic interoperability issues. MDA is an approach to deal with the development of software and in this case Web Services. By using MDA principles in the development of Web Services we aim for an approach to deal with semantic interoperability issues among Web Services, but before we go into the details of the our approach in chapter 7 we need an understanding of the technologies involved.

4.1 Web Services

Web Services has become another of the many buzzwords circulating the IT community. While Web Services haven't been around that long, more and more companies are starting up Web Service development projects. If they haven't already started projects, many are looking into the technology trying to find out how it may benefit their company. This section will give an introduction to Web Services and Web Service compositions, and show the semantical challenges that arise when developing Web Service compositions. We will also present two existing approaches to Web Service composition.

4.1.1 What are Web Services.

"A Web Service is a set of related functionalities that can be programmatically accessed through the web." [MBE03]

Web Services is a technology intended to make functionality accessible through the web independently of the programming language used. This makes it possible to use services made by others in your own software projects. Outsourcing services in this manner has a significant role in businesses ability to produce and deploy their own business solutions without having to invent the wheel a second time. Web services are programming language independent as an effect of their use of XML. XML is the same whether you are developing i Java, one of Microsoft's technologies or something else. This increases the interoperability, but still the semantics need to be dealt with. Even if you can access a Web Service independently of the implementation technology, you need to understand what the service does, i.e. the semantics of the service and the information it produces will have to be clearly stated in order for you to know that it is the Web Service you need for your project.

There are two types of Web Services, simple Web Services and composite Web Services. Simple Web Services are services that do not rely on other services to complete their task, and composite Web Services are services that do just that. The possibility to create new value-added Web Services or applications from a composition of other Web Services is one of the greatest advantages of Web Services. However, in order to create a composition of Web Services, the semantics of the Web Services in the composition needs to be understood. For a composition to work, what the Web Services do in terms of the information they need and the information they produce needs to be defined clearly, so that the Web Services can work together in performing some task.

Web Services strong relation with XML and XML-based standards is reflected by the definition used by the W3C Web Services architecture group;

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." [W3CWSA]

4.1.2 The Web Services standards.

There are three core standards that make up Web Services, SOAP, WSDL and UDDI, which are all based on XML. This section will provide an introduction to these along with some other standards that are important for the evolution of Web Services. Since Web Services is a relatively new concept, the standards are still evolving and has lead to interoperability problems among Web Services that use different standards. An initiative by corporate vendors called (WS-I) was set up to tackle this situation by recommending what standards to use. Their recommendation is called WS-I Basic Profile 1.0a and consists of the combination of SOAP 1.1, WSDL 1.1 and UDDI 2.0 [WSI03].

It is important to note that it is not necessary to make use of these standards to create something that can be called a Web Service, nevertheless focusing on standards and the evolution of them is important to ensure interoperability among Web Services.

4.1.2.1 SOAP

"SOAP Version 1.2 provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized, distributed environment." [W3CSOAP]

SOAP (SOAP used to be an acronym for Simple Object Access Protocol, but W3C no longer defines it as such) is the message exchange standard for Web Services, it is above http, ftp and smtp on the protocol stack. SOAP is built using XML and consists of an envelope with a header and a body. The purpose of SOAP is to exchange structured and typed information between peers in a decentralized and distributed environment. One of the advantages of using SOAP over http, which currently is the most used transport protocol for SOAP, is that it will not be blocked by any firewalls. SOAP is located on the communication level of the interoperability levels in section 2.1.1.

4.1.2.2 Web Services Description Language (WSDL)

"*Web Services Description Language (WSDL) provides a model and an XML format for describing Web services. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered*." [W3CWSDL]

WSDL is a document that specifies how the messages to and from a Web Service are structured, in other words it describes how to interpret the content of a SOAP message. In addition to this a WSDL document also specifies the address where the Web Service is located. WSDL documents contain messages which defines input and output using XSD datatypes. These data descriptions are on the application level of the interoperability levels in section 2.1.1.

A WSDL document consists of the following elements:

<definitions>	Container for the two categories of WSDL components, type system components and WSDL components. Type system components are defined by the <types> element. WSDL components are defined by <message>, <portType>, <binding> and <service> elements.
<types>	Describes the data types used by a Web Service using XML schema.
<message>	Describes the the abstract format of a particular message that a Web Service sends or receives. Can contain the <part> element.
<portType>	Describes a set of messages that a Web Service sends and/or receives by grouping related messages using the <operation> element. Renamed <interface> in WSDL 1.2.
<binding>	Describes a concrete binding of a <portType> and associated <operation> elements to a concrete message format and transmission protocol.
<service>	Describes one and only one interface that a service provides, and the endpoints it is provided over. Contains <port> elements.
<part>	Describes a portion of a particular message that a Web Service sends or receives.
<operation>	Describes an operation that a given <portType> supports. Contains <input> messages, <output> messages and/or <fault> messages.
<input>	Describes an input message.
<output>	Describes an output message.
<fault>	Describes a fault message. Expanded to <infault> and <outfault> in WSDL 1.2.
<port>	Defines the particulars of a specific end-point at which a given service is available. Renamed <endpoint> in WSDL 1.2.

Table 1 - WSDL elements

4.1.2.3 Universal Description, Discovery and Integration (UDDI)

"*The Universal Description, Discovery and Integration (UDDI) specifications define a way to publish and discover information about Web services.*" [UDDI]

While SOAP and WSDL defines how to communicate with a Web Service, UDDI intends to be the primary source for where to find the Web Services you need. UDDI is an industry-wide initiative to support Web Service discovery across technologies and platforms. It is a public registry which contains information about businesses and their web services. It is organized to serve as an online phone directory with white pages, yellow pages and green pages. The white pages lets you find

address, contacts, etc. of a specific business, the yellow pages lets you find businesses that are categorized according to some taxonomy, the green pages lets you find technical information about services that the registered businesses provide [UDDI].

4.1.3 Web service compositions and semantics.

Composing Web Services into new value-added Web Services or applications involves two phases, the discovery phase and the composition phase. The discovery phase involves finding the Web Services you need in order to create a composition, and the composition phase is where you deal with any discrepancies that exist in order for a Web Service to use the information produced by another Web Service.

4.1.3.1 The discovery phase

Web Service discovery is the name given to the process of finding a Web Service that is published in some form of registry or repository. This process involves the translation of the requirements you have for the Web Service you are looking for into a form that makes it possible to identify the Web Services that satisfies the requirements based on the descriptions of the Web Services that are already published. The discovery process can occur at three points in the lifecycle of software that makes use of Web Services, at design time, at run-time or at a point where the software is being expanded or upgraded.

- **Design-time discovery**

  Design-time discovery is discovery in the initial development phase, where the developers have possibilities to refine their queries in order to find the best suited Web Services. At design-time, the developers also have the possibility to test the Web Services they discover to make sure that they actually do what the descriptions say that they will do. Design-time discovery relates especially to specific purpose systems.

- **Run-time discovery**

  Run-time discovery is when the software itself realizes that it needs some information from a Web Service and automatically does the search of registries and repositories looking for services that can be used in order to obtain the missing information. A typical scenario in which this situation will arise is when a task is not performed because the Web Service that is supposed to perform it for some reason is not available. In addition to finding a Web Service that comes up with the missing information, the software will also have to generate mappings to account for any semantical or syntactical differences. General purpose systems will have to make heavy use of run-time discovery.

- **Upgrade discovery**

  Upgrade discovery is discovery done by the developers after the software has been deployed. A typical scenario for this is for instance if a Web Service doesn't perform as expected in terms of non-functional characteristics. In all other ways upgrade discovery is the same as design-time discovery.

The semantical challenge for the Web Service discovery process is to match the descriptions of the requirements to the descriptions of the Web Services that are available.

4.1.3.2 The composition phase

The composition phase is where all the Web Services are put together to create Web Services with added value or just conventional applications making use of Web Services. In order to this the order of the Web Services in the composition needs to be known so that you know what Web Services will use information from other Web Services. When the order is known semantical, syntactical and structural conversions will have to be developed to make sure that the information is in a form that will be understood by the Web Service using the information. You also have to know if the Web Services require that their individual operations are invoked in a particular order. In addition to this, strategies concerning what to do in case of the failure of a Web Service have to be developed. Failure in one service can break down the whole composition, which is a situation to be avoided.

The Western Development Laboratories (WDL) have developed a composition theory which is a mathematical theory for modelling message passing systems. The WDL composition theory has been applied to analyzing security architectures in [WDL], and we present here some of the elements of WDL composition theory presented in that analysis. Models for components and the system are represented graphically by circles or "bubbles". These component and system models are black-box models, thus identifying the boundary of the component or system. Relationships between components are represented by lines connecting the "bubbles". Components and their functionality are denoted with uppercase letters, A, B, C, etc. S denotes a system. Components have two kinds of composition, serial and parallel. Serial composition is denoted by $A \circ B$ and parallel composition is denoted by $A \cup B$. Serial composition means that the output from A is input to B. Parallel composition means that outputs from A and B are combined. Figure 8 shows serial and parallel composition.

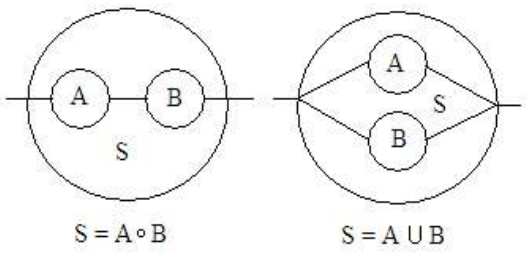

Figure 8 - Serial and parallel composition

In terms of Web Service compositions the Web Services involved are the components and the composition is the system. The WDL composition theory can be used to analyze the Web Service composition with respect to several aspects, such as assessing changes in functionality and security policy due to changes in system architecture. Our intention of introducing the WDL composition theory here is to provide a vocabulary for discussing Web Service compositions.

With the aim of dealing with Web Service compositions several standards have emerged such as Business Process Execution Language for Web Services (BPEL4WS), Web Service Choreography Interface (WSCI) and Business Process Modelling Language (BPML). These standards are often called orchestration or choreography standards. Orchestration standards are intended to support the business processes. "*Web service orchestration is about providing an open, standards-based approach for connecting Web Services together to create higher-level business processes"* [P03]. In relation to the different levels of interoperability presented in section 2.1.1, these standards support interoperability at the business level. Table 2 shows the Web Service stack and where semantics is an important issue.

BPEL4WS	Service flow and composition	Semantics
Trading partner agreement	Service agreement	
UDDI / WS Inspection	Service discovery (focused and unfocused)	
UDDI	Service publication	
WSDL	Service description	
WS Security	Secure messaging	
SOAP	XML messaging	
HTTP, FTP, SMTP, MQ, RMI over IIOP	Transport	

Table 2 - Web Service stack and semantics [BFM02]

4.1.4 Web services and ontologies.

Web Services can benefit from ontologies in several different ways. First of all, the parameter and return values of the operations of a Web Service needs to be clearly understood by the ones using them. This is a case of understanding the data involved, and in this case ontologies can be helpful. Secondly, in the process of finding a Web Service that fits some set of requirements, there needs to be an understanding of what the requirements and what the services can offer are, so that a matching can be done.

The DARPA Agent Markup Language program have developed an upper-level ontology of services which is called DAML-S. DAML-S is made up of three main parts; the service profile for advertising and discovering services, the process model, which gives a detailed description of a service's operation, and the grounding, which provides details on how to interoperate with a service via messages. DAML-S provides a way of matching requirements with descriptions of Web

Services by letting them be described in a similar manner, thus enabling automated reasoning. The input and output of Web Services can in DAML-S be described by DAML+OIL which makes it possible to determine the relationship between input and output of a requested Web Service and input and output of an existing Web Service.

DAML-S is an initiative following the ideas behind the Semantic Web, and is built on top of DAML+OIL which is built on top of RDF/XML, and future versions of DAML-S are likely to be built upon OWL when it becomes sufficiently mature [DAMLS], and will preliminary versions are renamed OWL-S. A more detailed presentation of DAML-S is given in section 4.3.2.

4.2 Existing approaches to Web Service compositions.

Web Service composition is dominated by two approaches, the standards approach and the DAML-S approach. By the standards approach we mean those with a foundation in the three core Web Service standards, SOAP, WSDL UDDI, and associated with an orchestration or choreography standard. The DAML-S approach was originally intended for the more dynamic agent-based world, but also apply to specific purpose systems. Within these approaches there are many variants on how to deal with Web Service compositions, but we will here only present the general aspects indicating the differences and similarities of the approaches.

4.2.1 The standards approach to Web Service composition.

UDDI is the the standard for Web Service discovery in the standards approach. The UDDIs green pages is intended to hold all technical details of a businesses services. The idea behind UDDI is that you have a standardized way to let the world know that you have a Web Service. The world just have to go to the UDDI and find your Web Service. To find a Web Service by searching a UDDI, you can search in the different categories of businesses, or you can search for a service more or less the way you would search for information on the Internet, i.e. by providing one or more keywords.

A UDDI registry can be searched by businesses, services and service types. In this context searching for services and service types are most interesting. These can be searched for by a combination of service name or service type name and category [UDDI]. In order to support run-time discovery the UDDI registry contains bindingTemplate data providing details necessary to invoke a Web Service.

WSDL is the standard for describing the processes needed to utilize a service, and therefore every Web Service has a WSDL document associated with it. It describes the service parameters and the transport protocol, but what the service does is left to interpretation using the service name and the parameter-names, -types and -restrictions, which are described by referencing to XML Schema Definitions (XSD).

Among the different standards for dealing with compositions we have chosen to look at BPEL4WS for the standards approach. BPEL4WS is an initiative by corporations such as IBM and Microsoft, which defines a notation for specifying business process behavior based on Web Services

[BPEL4WS]. BPEL4WS can describe business processes in two ways, executable processes and abstract processes. Executable processes model the actual behavior of a participant in a business interaction. Abstract processes describe business protocols which specify the mutually visible message exchange behavior of the parties involved without revealing the internal behavior. The BPEL4WS process is modelled on top of WSDL.

A business process described by BPEL4WS, uses the constructs in table 3 to define the process.

<receive>	Allows a business process to do a blocking wait for a matching message to arrive.
<reply>	Allows a business process to send a message in reply to a message that was received through a <receive>
<invoke>	Allows a business process to invoke an operation on a WSDL portType offered by a partner.
<assign>	Lets the values of data in containers be updated with new values.
<throw>	Generates a fault from inside the business process.
<terminate>	Allows you to immediately terminate a business process.
<wait>	Allows you to wait a given time or until a certain time has passed.
<empty>	Allows you to set a “no-op” instruction into a business process.
<sequence>	Allows you to define a collection of activities to be performed sequentially in lexical order.
<switch>	Allows you to select exactly one branch of execution from a set of choices.
<while>	Allows you to indicate that an activity is to be repeated until a certain success criteria has been met.
<pick>	Allows you to block and wait for exactly a suitable message to arrive or for a time-out alarm to go off.
<flow>	Allows you to specify one or more activities to be executed in parallel. Links can be used within parallel activities to define arbitrary control structures
<scope>	Allows you to define a nested activity with its own associated fault and compensation handlers.
<compensate>	Is used to invoke compensation on an inner scope that has already completed its execution normally.

Table 3 - Constructs defining processes in BPEL4WS, from [BPEL4WS]

4.2.2 The DAML-S approach to Web Service composition

"*DAML-S supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form. DAML-S markup of Web services will facilitate the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring. Following the layered approach to markup language development, the current version of DAML-S builds on top of DAML+OIL.*" [DAMLS2]

The top level of the service ontology consist of the class SERVICE has the properties *presents*, *describedBy* and *supports*. The respective ranges of these properties are the classes SERVICEPROFILE, SERVICEMODEL and SERVICEGROUNDING. The service profile answers the question, what does the service do? The service model answers the question, how does the service work? The service grounding answers the question, how is the service accessed? [DAMLS]

For Web Service discovery DAML-S lets service be described by a service profile which describes a service in terms of its inputs, outputs, preconditions and effects (IOPE). The IOPEs can be represented with restrictions on their values and a reference to the corresponding IOPEs in the process model. The IOPEs are currently described by DAML+OIL, but are likely to be described by OWL in the future, the point is that they are described by an ontology markup language.

The process of discovering a Web Service with DAML-S is by creating a profile description of the Web Service you need and the let either a registry or an agent compare the profile of the requested service with the profiles of the services that are available and return the matching Web Services.

For the composition phase DAML-S provides the Service model and the service grounding. These parts corresponds to WSDL where the service grounding describes how to access the Web Service and the Service model describes the service itself. The service model has a subclass which is called a process model, which views a Web Service as process. This subclass describes all IOPEs of the Web Service in DAML+OIL so that it is possible to use automated reasoning on them, and therefore make it possible to use them in compositions as their semantics are described. In contrast to BPEL4WS, the process model is only descriptive. The process model is made up of the constructs in table 4.

AtomicProcess	Processes that are directly invocable, have no subprocesses and execute in a single step, from the perspective of the service requester.
SimpleProcess	Processes that are not invocable, but are conceived of as having single-step executions. Used to either provide a view of an atomic process or as a simplified representation of a composite process.
CompositeProcess	Processes that are decomposable into other (noncomposite or composite) processes. Specified using the control constructs below.
Sequence	A list of processes to be done in order.
Split	A bag of processes to be executed concurrently.
Split+Join	A bag of processes to be executed concurrently with barrier synchronization.
Unordered	A bag of processes to be executed in some unspecified order, or concurrently.
Choice	Control construct used to chose among a set of processes.
If-Then-Else	If test condition is true, do then-process, else do else-process.
Iterate	Iterate over a process.
Repeat-Until	Repeat process until some termination condition holds.

Table 4 - Constructs for defining processes in DAML-S, from [DAMLS]

4.3 Model Driven Architecture (MDA)

Model Driven Architecture (MDA) is an initiative from the Object Management Group (OMG) that aims to be an *"open, vendor-neutral approach to the challenge of business and technology change."* [OMGMDA]. In 2002 the OMG announced MDA as its strategic direction. In the nearly two years since then, many discussions have taken place about what MDA is. This section will reflect our view on MDA with focus on elements that can be useful for creating Web Service compositions.

4.3.1 What is MDA.

The main idea behind MDA is that business or application logic should be separated from the underlying platform technology. By doing this you will be able to develop applications independently of platforms and use code generation techniques to realize your application on a chosen platform. If technology advances or other changes to technology forces you to change your chosen platform, you will have captured the business or application logic independently of your platform and thus make you able to use code generation techniques again to realize your application on the new platform.

MDA is based upon several key standards of the OMG, these include Unified modelling Language (UML); Meta-Object Facility (MOF); XML Meta-Data Interchange (XMI); and Common Warehouse Meta-model (CWM) [OMGMDA]. These standards are there to facilitate the idea of *"using modelling languages as programming languages rather than merely design languages."* ([MDAF03], p xv). The same idea has been used for years to generate real-time and embedded systems. MDA is a term for doing this in an enterprise software systems context. Figure 9 shows OMGs overview of MDA.

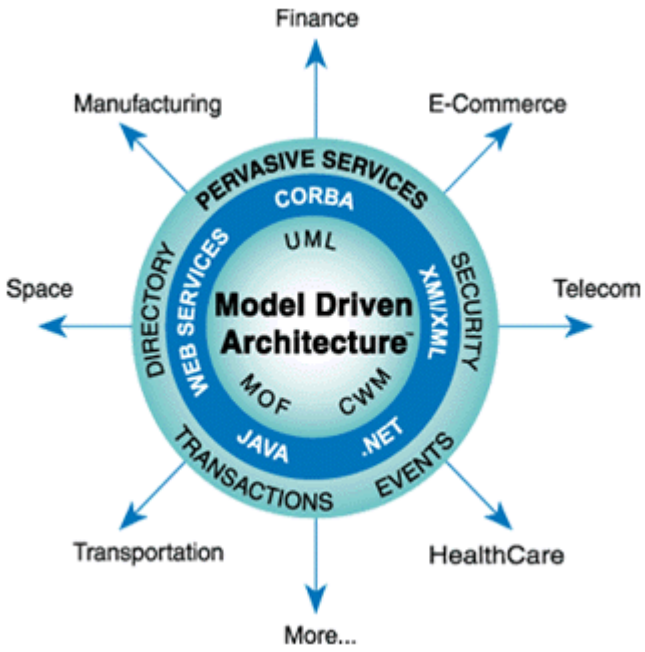

Figure 9 - MDA overview [OMGMDA]

The key standards are at the core with different platforms as the next layer illustrating the step away form platform specific details. In terms of a development process, developers will start at the core by capturing business or application logic by using UML models, then use generation techniques to add platform details necessary to generate the code that will realize the application. While there are many obstacles that must be overcome for the vision of MDA to be fully realized, the situation today is that there are new tools coming on the market that supports generation of platform code leaving only business logic code to be dealt with by programmers. This situation provides a step in the right direction as programmers can focus on the "interesting" code and leave the tedious platform code to generation tools.

4.3.2 Key MDA concepts.

MDA defines some key concepts that are vital to enable the vision of MDA. The most important key concepts involve the use of UML models as programming languages, as this provides an abstraction from third and fourth generation programming languages, by generating code and

models from models.

- **Abstraction.**

  Using a modelling language as programming language provides yet another level of abstraction in the world of programming. This abstraction can be seen as the next level in the evolution from machine-code to assembly to third and fourth generation programming languages. The raising of abstraction level provides improvements to development productivity as well as the quality and longevity of the software that is created, by relieving the programmer of platform dependent coding and letting him or her focus on the business or application logic.

  Using a modelling language also makes it possible to model at different levels of abstraction, such as a business model, a requirements model, a model without technology details and a model with technology details. While there is not much new with this, MDA aims to tie the models and the running code tighter together by generating models and code from models.

  Essential concepts within MDA is the separation into platform specific models (PSM) and platform independent models (PIM). In the PIM all details depending on a specific platform is abstracted away. MDA does not define which platform you are abstracting away from, this has to be decided by the developers, this leads to the perhaps confusing notion that a PIM can at the same time by a PSM, just with regards to a different platform. However it is common to refer to a model as being independent or specific to middleware technologies such as CORBA, .NET and J2EE.

  The PIMs primary function is to ease the communication among humans aiding in the process of understanding the system that is going to be developed. In this process the focus is on understanding the business logic and any technological details only serve to confuse and take away the focus. A PSM on the other hand is a model of how the system can be realized on a specific platform. The PSMs function is to communicate to the programmer and to be the base for code generation.

- **Code and model generation.**

  The real strength of MDA as many argues is the promise to generate code from models and vice versa, making it possible to program using design languages such as Unified modelling Language (UML). In addition it will also be possible to generate models from models, e.g. a PSM from a PIM. While the ability to generate code today is rather limited, other areas such as real-time and embedded systems already generates most of the code. The ability to generate code is dependent on formal models which are semantically rich. UML models cannot convey all the semantics needed and therefore extensions such as Object Constraint Language (OCL) and Action Semantics have been added. OCL lets you define constraints in a formal way and Action Semantics supports precise executable specification of systems.

  In the MDA tools that are starting to emerge, code generation is limited to generate skeleton classes where the programmer must add code supporting the business logic in so called free-blocks, which are areas in the generated code that the programmer is allowed to edit. The most

demanding challenge that MDA is facing is letting the "hand coded" business logic be reflected in the models the code is generated from. The synchronization of models and code, where any changes in the code is reflected in the models and vice versa, is known as round trip engineering. The ability to do this is by many regarded as the critical success factor for MDA, and has therefore lead to much skepticism as this is seen by some as an impossible task.

4.3.3 MDA and interoperability.

The concepts of PIM and PSM are also interesting seen from a semantic interoperability view point. When trying to understand the semantics of another system, technological details are uninteresting, they say little or nothing about what the system does, the semantics of the data/information the system needs or produces, or which parts of the system does what. In such cases, human interpretations might be needed, and this is where PIMs come into the picture as they are all about communication among humans. A clear model of a system focusing on the business logic can give the developer a much clearer understanding of the system and thus help him/her to make his/her system interoperable with this system.

4.3.4 MDA and Web Services.

In the light of MDA, Web Services is only another platform, and modelling Web Services could be done by at first abstracting away from the Web Service specific details. The Web Services design vocabulary should let us describe the information and services in ways that are entirely independent of XML, WSDL, SOAP, UDDI, Java and other Web Service implementation technologies. The trend is towards tools that can automate production of XML, WSDL, SOAP, UDDI, and artifacts and implementation code from design input [MDAF03].

4.4 Summary.

This chapter introduced Web Services and MDA. Web Service were introduced because of the interoperability issues arising when developing Web Service compositions, which is the focus of this thesis. We presented two different approaches for dealing with Web Service compositions, the standards approach and the DAML-S approach. Finally we introduced MDA, which contains aspects concerning modelling that we will make use of in chapter 7 where we present our approach to Web Service composition.

5. An example case of Web Service Composition

This chapter describes an example case used in this thesis to identify the requirements for creating Web Service compositions in chapter 6, and used to illustrating our proposal, MOSIS, in chapter 8. The case example is called MODUSA (MOdel Driven User oriented Service Architecture) and was developed in cooperation with three other students, each with a different responsibility corresponding to the respective theses. MODUSA was modelled as a Web Service and implemented as a client that uses four different Web Services to show on a map how a gas plume will disperse under the current atmospheric conditions after a chemical accident.

The chemical dispersion case we use is adopted from the ACE-GIS project, which is described in the next section. The case example is described using an MDA approach based on the concepts PIM and PSM.

5.1 ACE-GIS

ACE-GIS (Adaptable and Composable E-commerce and Geographic Information Services) is a research project of the Information Society Technologies Programme of the European Union [ACEGIS].

The partners of the project are:

- *Norwegian Mapping Authority (Norway)* – Administrative coordinator and responsible for the Environmental Pilot Demonstrator
- *Ionic Software (Belgium)* Exploitation Manager and responsible for GI services
- *e-blana (Ireland)* – responsible for EC services and the Emergency Pilot Demonstrator
- *University of Münster (Germany)* – responsible for Semantic Interoperability and interoperability architecture
- *University of Jaume I (Spain)* – responsible for conformance testing tools, standards integration and dissemination
- *SINTEF (Norway)* – Technical coordinator and responsible for open source model transformation toolkit
- *INESC-ID (Portugal)* – responsible for Open Source Composition and workflow services

The objective of the ACE-GIS project is to provide a set of tools and a service infrastructure that enable:

- Developers to efficiently build web services, integrate existing services from multiple sources, and compose services to form new value-added compound services
- Service providers to register services, monitor service use and regulate contracts with users
- Users to discover, access, configure and use available geographic information services through adaptable applications.

With this objective in mind students from SINTEF have realized a small scenario of use and development of web services addressing some of the issues being raised in the ACE-GIS project.

5.2 The MODUSA gas dispersion web service.

The MODUSA gas dispersion web service is a web service that will, given the location of a chemical accident, provide a fire officer with a map of the area with a gas plume superimposed on it. The gas plume shows the area affected by the chemical release. This information will give the fire officer a quick way of determining what areas needs to be evacuated. Today there exists applications that can do this, but they often require the fire officer to provide lots of information concerning the chemical and the weather. In times of emergency the fire officer will want to provide as little information as possible and receive the information he needs as quickly as possible. To sum up, the goal of the web service is to provide enough information for the fire officers to initiate the evacuation in the most exposed areas first, with a minimal of information gathering by the fire officers. The way we have dealt with this problem is to develop a web service that is composed of four other web services which will provide weather data, chemical data, a map and a gas plume estimate. Putting the responsibility of providing information on web services instead of the fire officer, gives not only a quicker response to the officer, but it also makes it an easier system to use.

The web service is a composition of four web services, a weather service, a chemical properties service, a gas dispersion service and a web map service. The stereotype businessService [COMET] was used to indicate that the involved systems were services. This is shown in figure 10.

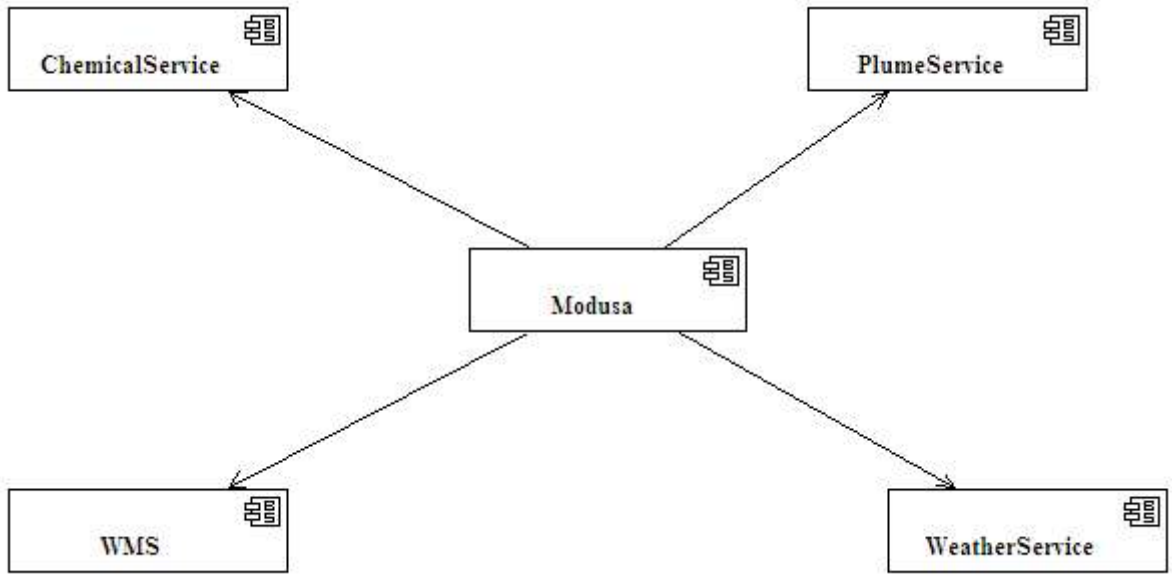

Figure 10 - The Web Services involved in the MODUSA case example

The web services involved in our example are described below.

- **The weather service**

  The weather service gives the weather information needed to make the estimate of the gas dispersion as accurate as possible. In this example the only weather information needed is wind speed and wind direction. In real life you would want to take more factors into consideration, but for this example the accuracy of the estimation is not the focus. The weather service we have used in this example is developed by Cape Science, and gives weather information for all airports in the world that have an ICAO code.

- **The chemical information service**

  The chemical properties service, gives the chemical properties of a given chemical, which in real life also would be taken into consideration when calculating the gas dispersion estimate. This example case does not use the chemical information for calculating the estimate. The chemical service will also given the location of an accident provide chemical information about the chemical(s) stored in that location. This web service is developed by us and is only a dummy service mimicking how a chemical information service could function if there existed any.

- **The gas dispersion plume service**

  The gas dispersion service calculates an estimate of how far the chemical might spread given the current weather conditions and the emission rate of the chemical. The service returns an image of a gas plume, which can then be superimposed on a map of the area. This service is developed by

IONIC Software as part of the ACE-GIS project.

- **The web map service**

  The web map service gives a map of the given area. This service is developed by Statens Kartverk and only gives maps of areas in Norway.

Figure 11 shows a sequence diagram of the order in which the MODUSA Web Service invokes the other services.

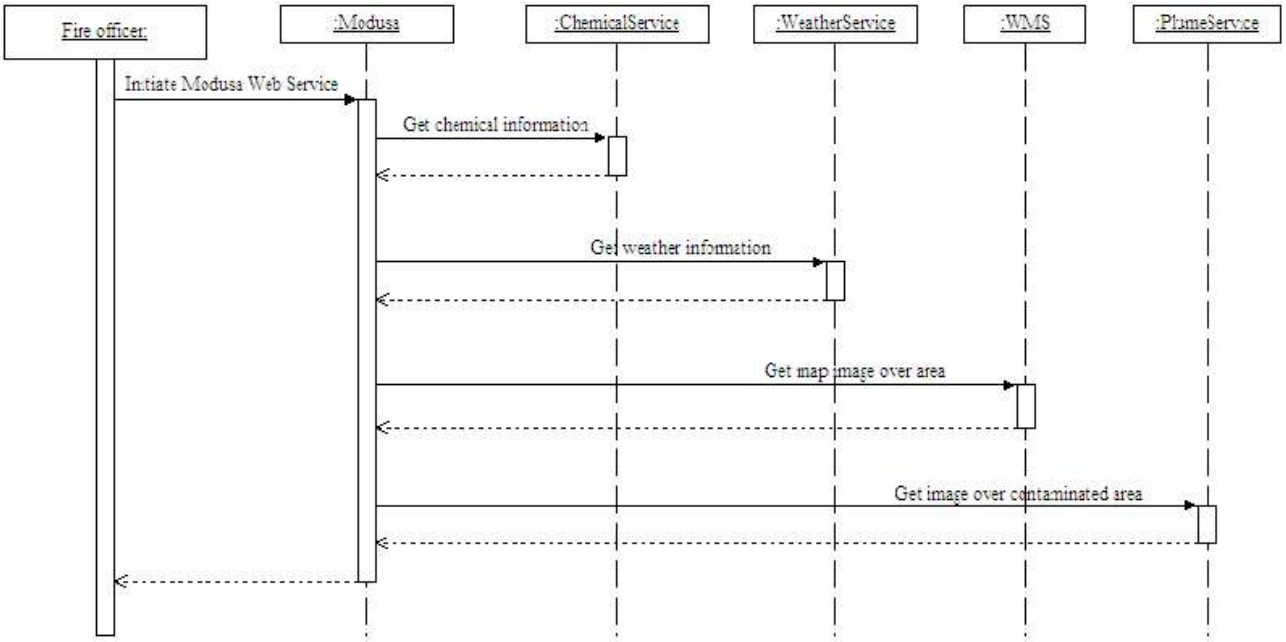

Figure 11 - The sequence of the Web Services in the MODUSA case example

The important aspect for the sequence is that the Ionic Gas Dispersion service is invoked last, after the information needed is provided by the other web services.

Figure 12 shows an example of how a client might use the web service. This is a screen shot of a client we initially developed to familiarize ourselves with the technology, which uses the four Web Service directly instead of going through the MODUSA Web Service.

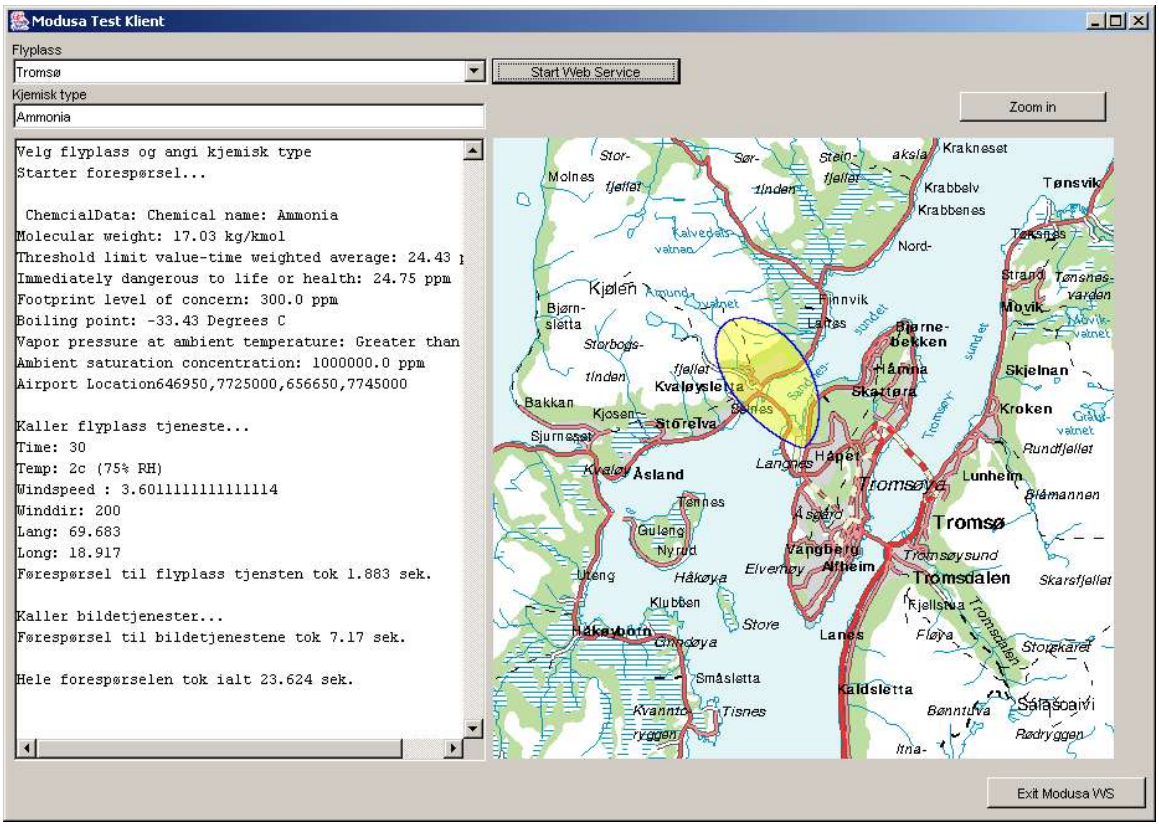

Figure 12 - Screen shot of the MODUSA case example client

5.2.1 Limitations

As a consequence of the immaturity of the web services technology, our gas dispersion web service has some limitations which comes from the fact that currently there are not many web service available. Most web services that you can find today are in the experimental phase, showing the direction the evolution might take. The limitations of our gas dispersion web service are as follows; it gives an estimate of the gas dispersion for chemical accidents in airports in Norway.

5.3 Platform Independent Model (PIM) of the MODUSA Web Service

This section is a description of the PIM and our considerations connected to it and the modelling process.What we wanted to emphasis in the PIM was the general functionality within the application. Our focus was to model the functionality in a manner that could be represented throughout variuos types of environments e.g. J2EE, .NET and different languages within these platforms. Consequently the PIM will be used as a context model for our application. This meant that we had to disregard elements that could be seen as environment specific and instead focus on more general concerns.

Other elements that were important was how to best model and solve application interchanging concerns. How was our Web Service best to communicate with others? And what type of signatures and interfaces were to be used? Since the MODUSA web service uses other web services we needed to investigate such problems already at the PIM level. We solved these problems by having a specific connector for each service involved. The connectors take care of how to connect to the services and how to deal with the information provided by the services. The MODUSAImpl class takes care of the ordering of the Web Service Calls and the building of the information that the Web Service will return. The services are modelled as business services here to avoid platform specific details. The services don't necessarily need to be Web Services.

Another issue was how to best model for changes in the external web services. By changes we mean replacements by new web services or changes in the existing web services. This we have done by introducing implementation specific classes which implements our static interface. This might be realized by one implementation per web service, as in our example, or by a controller class that is responsible for chosing among available web service implementations which perform the same task. This is very useful for critical services that cannot be dependant of another web service breaking down. Figure 13 shows a UML class diagram of the PIM of the MODUSA Web Service

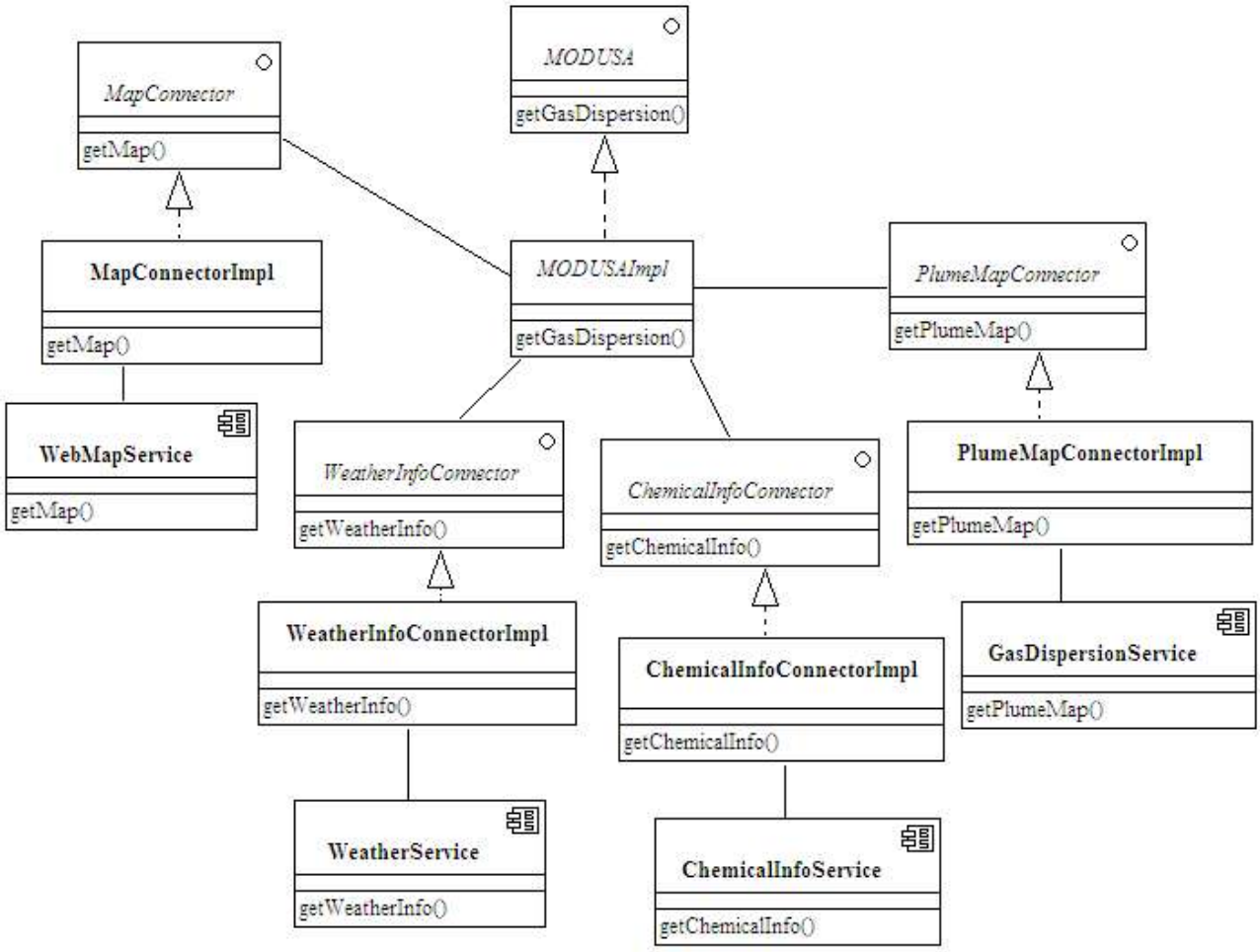

Figure 13 - The PIM of the MODUSA case example

5.4 Platform Specific Model (PSM) of the MODUSA Web Service

The next phase in the development project of MODUSA was to build a PSM on the basis of our PIM from the previous level. Figures 14 and 15 show the PSM of the MODUSA Web Service. The figures represent the most important aspects of the PSM as the complete diagrams are too large to be useful here. The figures show the chosen Web Services with their signatures. We have also included the internal data formats that the specific connectors need to convert the information from the Web Services to. The services are now modelled as Web Services, reflecting that platform specific details are included.

Figure 14 shows the Web Map Service from Statens Kartverk and the weather service from CapeScience. The respective connectors hold the information on how to connect to the services and how the information from them will be converted to the internal data format of the MODUSA Web Service. If, for instance, an alternative service providing weather information is found at a later point, the weather service from CapeScience can be replaced. Doing this only involves creating a new connector that implements the WeatherInfoConnector interface.

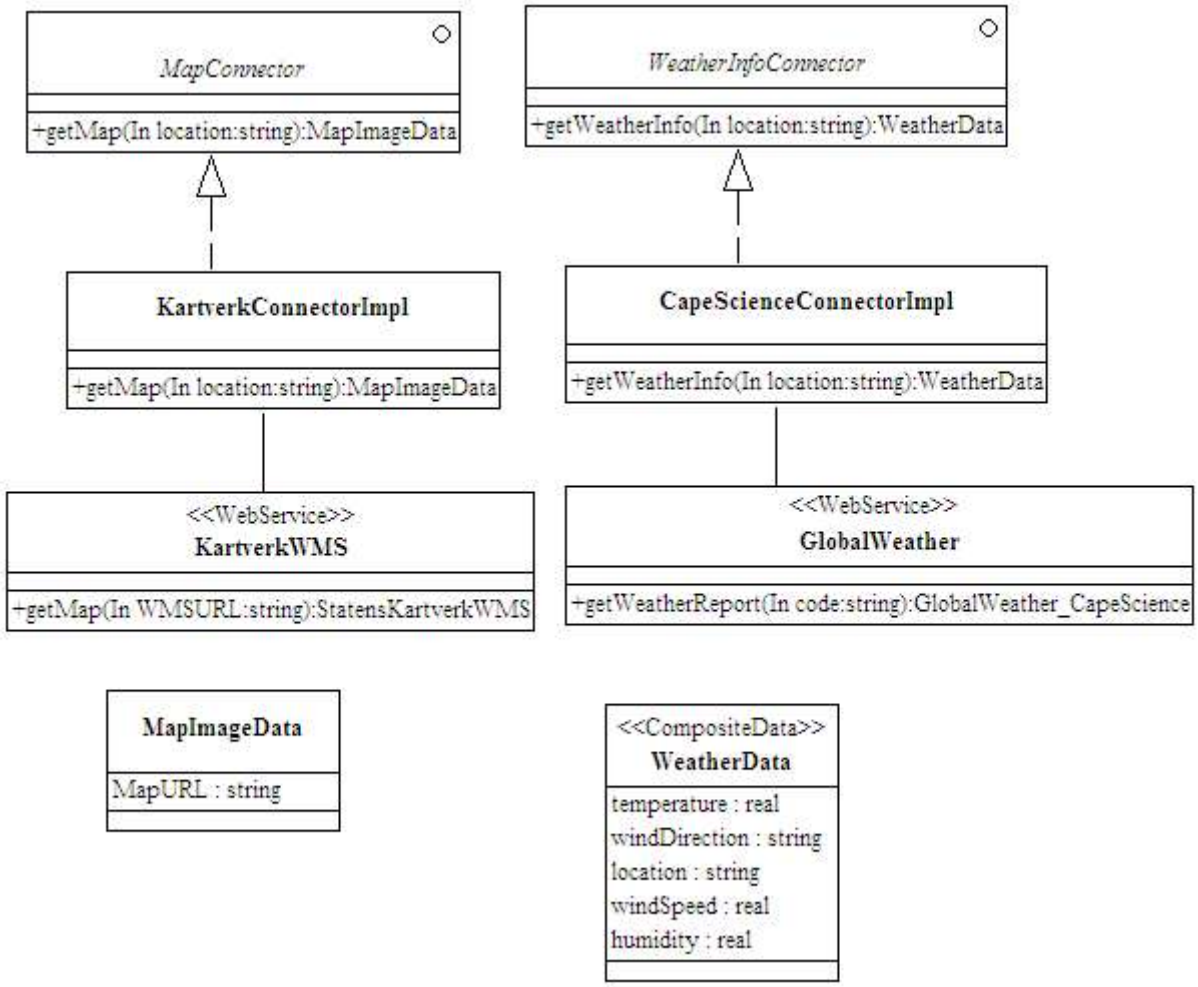

Figure 14 - Part 1 of the PSM of the MODUSA case example

Figure 15 shows the chemical information Web Service we created as part of this project and the gas dispersion service created by Ionic as part of the ACE-GIS project.

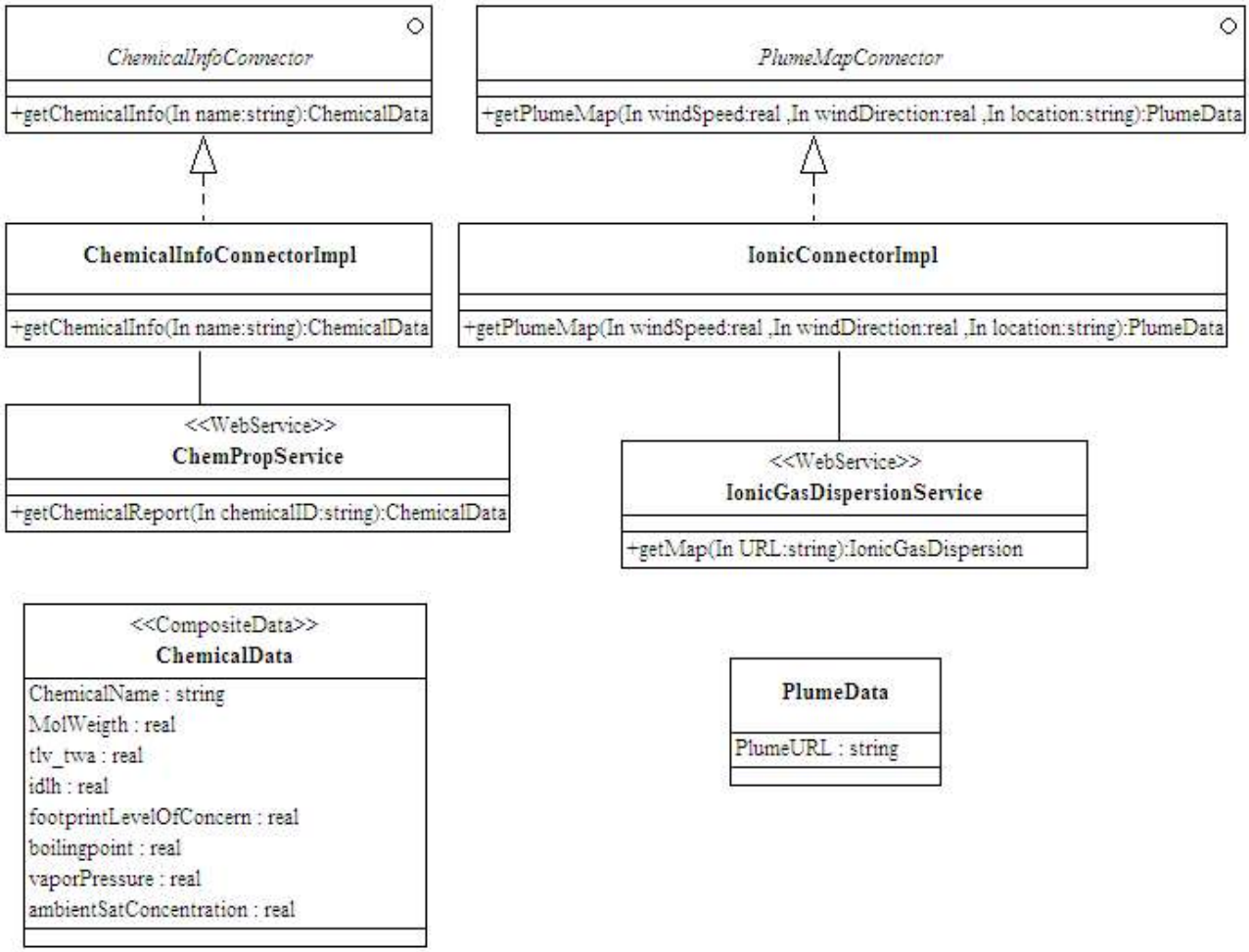

Figure 15 - Part 2 of the PSM of the MODUSA case example

Changing the focus to the PSM level signifies a more detailed approach in the model work. The two PSM figures also visualize how the models now incorporate a much higher specification level. From each Web Service connector class a belonging data class is associated. This approach of data structure modelling liberates us from the data structure of the external Web Service as we then are able to build our own complex objects based on the incoming data, which is not directly coupled with the exterior structure.

5.5 Reflections on the MDA approach

The development process for the MODUSA case was based on a MDA approach. Our main goal was to test the method process of MDA and see how it influenced our work. The principal interests were if the model-driven elements gave us any significant advantage or benefit in contrast to other approaches.

We invested a great amount of time and effort to resolve the domain model, in the start-up phase of the project. Much of the preliminary work was based on getting to know the problem area and understand the different challenges connected to developing the challenges. Next, designing and modelling the PIM was mainly the major element during this phase. Problems related to how to best construct and build the application was still the main problem, at this stage of the development process. Issues on how to best solve the problems of handling various data structures form the other applications were another aspect that we had to take into consideration at this stage in the development process.

Both of the problem areas stated above were issues related to the PIM level, and thus handled in the early stages of the development phase. The PIM model were subject to several iterations, which gave us the opportunity to repeatedly address several modelling related issues. To omit technical details at this level where definitely an advantage because it enabled us to focus on better functionality and not technical problems. Several iterations at the PIM level also gave us an advantage when we advanced to the PSM level. More and more technical issues emerged during the finishing parts of the PIM specification, which then became the base of our approach of the PSM level model. Data class and other elements of added functionality were already problems well known and we were therefore prepared for them, and some even solved as a direct result of the specification and guidelines laid out from the PIM level.

So what are the conclusions of the experiment with developing the MODUSA case based on the MDA framework? One of our findings was that several iterations on the same model level helped us to address difficult problems in a more expanded fashion. Iterations help you to address a specific problem and also releases you from addressing others at the same time. Another and important finding were that by omitting technical or platform specific issues, we were able to solve problems related to functionality instead. As a result of this, the main functionality was already in place when we started to construct data objects and other platform related issues.

5.6 Lessons learned from MODUSA with respect to semantic interoperability

The development of the MODUSA case example confirmed our beliefs that the discovery of Web Services is a challenging semantic issue. The fact that the only UDDIs that exist are for testing purposes, made the search even harder. The only Web Service we needed to discover was the weather service, since we already had the other ones. Lacking semantic matching engines we had to find weather service using conventional search engines, such as Google, and the semantic matching needed to evaluate the services we found had to be done manually.
When developing mappings between the Web Services we also got a hands on experience in the difficulty in understanding the complete meaning of the elements involved. Even if we found what elements corresponded to each other, understanding the complete meaning of the elements proved to be difficult. For instance the wind direction from the weather service was slightly different from the wind direction needed by the plume service. Without any descriptions of the elements it took quite a lot of trial and error before we figured it out.

5.7 Summary

This chapter presented the MODUSA case example were we created a client that makes use of a composition of four Web Services. This example is used to define the requirements in chapter 6 and to illustrate our approach to Web Service composition in chapter 8. The client was developed using MDA principles such as a platform independent model and a platform specific model.

Part II

–

MOSIS and how we got there.

6. Requirements

This chapter will explore the requirements put on any techniques that deal with Web Service compositions. Creating Web Service compositions that are semantically interoperable puts strains on both the discovering of Web Services and the composition it self. The semantics of the requested Web Service needs to be matched against the semantics of the Web Services that are available. In addition the semantics of the data that will flow between the Web Services in the composition will need to be the same for all the Web Services using the data. The requirements identified in this chapter are mainly from the development of the MODUSA case example, but also comes from other sources. This chapter is split into three logical parts, design requirements, discovery requirements and composition requirements. The conclusion of this chapter will evaluate the existing approaches to Web Service compositions from chapter 4 in terms of the requirements.

6.1 Design requirements

The design of a Web Service or application that is composed of other Web Services will need to be tolerant to external changes. When dealing with a distributed environment, there is no guarantee that the Web Services used will be up and running at all times. Of course, these design requirements only apply to Web Service compositions that are defined to only perform a specified task. Semantic Web agents with general problem solving abilities will face other requirements than being able to

tolerate changes in the Web Services they use as they will dynamically decide what services to use to solve the problems they are given. Web Service compositions set up to perform a specific task need to tolerate changes in order to perform their tasks. There are several possibilities of what to do in the event of an external change;

1. No tolerance to change, service is down.
2. List of equivalent services, if one fails try the next on the list.
3. Dynamic discovery of equivalent service or services.
4. Combination of 2 and 3, if all services in the list fail, try dynamic discovery.

In order to achieve interoperability it is not sufficient to be able to complete the task once, the Web Service will need to complete the task every time it is asked to complete the task. This brings us to the first requirement.

R1 – Change tolerance:	Web Service compositions that are set up to perform a specific task repeatedly will have to be tolerant to change in order to achieve interoperability.

6.2 Discovery requirements

The discovery of Web Services can happen at two stages in the lifecycle of a Web Service composition. It can be during the development stage or during runtime. Either way, the semantics of the descriptions of an existing Web Service has to match the semantics of the requirements defined that the requested Web Service has to fulfill. In this context requirements refers to both functional and non-functional requirements.

6.2.1 Functionality matching

During the design phase of a project a decision to have some specified task performed by a Web Service requires that you find one or more Web Services that, using the information you already have, come up with the information you need. In other words if a Web Service is of the form $\boldsymbol{Input \rightarrow Output}$, then the requested Web Service is of the form $\boldsymbol{Input_R \rightarrow Output_R}$, where $\boldsymbol{Input_R}$ is all the information that is available in order to obtain $\boldsymbol{Output_R}$, and $\boldsymbol{Output_R}$ is the requested information. All the Web Services that are available is a set **P** which is also of the form $\boldsymbol{Input_P \rightarrow Output_P}$. What you are after is the subset of **P** that satisfies the requirements of **R**. The subset $\mathbf{P_1}$, that produce the information you need, is the set of Web Services where $\boldsymbol{Output_R \subseteq Output_P}$ or where $\boldsymbol{Output_R}$ can be derived from $\boldsymbol{Output_P}$, and the subset $\mathbf{P_2}$, that produce something given the information that is possible to provide, is the set of Web Services where $\boldsymbol{Input_P \subseteq Input_R}$ or where $\boldsymbol{Input_P}$ can be derived from $\boldsymbol{Input_R}$. In this context derived can mean a simple transformation or finding a Web Service that with the information that is available can produce the information that another Web Service needs to produce $\boldsymbol{Output_R}$. The subset that fit the requirements is thus the intersection of $\mathbf{P_1}$ and $\mathbf{P_2}$. For example, imagine that you need a Web

Service that finds a given persons birth date, and the information you have available in order to obtain this information is the names of a group of persons. The subset $\mathbf{P_1}$ is then the set of all Web Services that will given some input produce a persons birth date, or something that can be used to obtain the birth date, for example a persons age in days. The subset $\mathbf{P_2}$ is the set of all Web Services that take as input the name of a person, or something that can be derived from the name, for example the persons telephone number. The intersection of these sets will give you a Web Service or a set of Web Services that together find a persons birth date given his or hers name.

Finding the right Web Service means that you have to present your requirements in the form ***Input → Output*** in a manner that lets it be semantically matched with the descriptions of the set of provided Web Services. This breaks the task down to how to describe a Web Service so that it contains enough information to make the matching against the requirements which can be described in the same manner. If it is possible to perform a search for Web Services using this approach, the semantic matching that has to be done will be the same as the semantic matching that has to be done when developing Web Service compositions. The only difference is that you'll have to match output against input or some internal representation against input and output against the internal representation.

When Web Services have been found that fit the requirements and has been tested to make sure they actually do what they are advertised to be doing, other selection criteria such as difficulty in developing mappings and non-functional characteristics can be used to choose among seemingly equal Web Services.

To find the right Web Services for a project you need a clear picture of the processes that your project under development is going to perform, all task need to be identified so that the ones that are tasks that can be performed by a Web Service have a clearly specified objective along with the data flow in terms of what information it will have available and what information it will be expected to produce. In other words you need to know what you want, and how you can realize it.

Our view on Web Service discovery is shared by [GYP02] and [PKPS02], where discovery is performed by functionality/capability matching.

R2 – Functionality matching:	Web Service discovery needs to be done by matching of functionality.

6.2.2 Describing the Web Service you need

Representing the requirements of a requested Web Service in a form that is interpretable by software involves finding the balance between human-understandable and machine-understandable. This balance point varies depending on the user, ranging from experts in machine-understandable description languages, such as ontology representation languages, to normal users with no knowledge of description languages or programming languages.

- **Expert in ontology representation languages**

  When this user needs a Web Service for performing a specified task, he or she will simply describe the requested Web Service and use that description to query a registry or repository of Web Service that is capable of matching based on functionality.

- **Software developer**

  This user is typically trained in modelling languages and or programming languages, and may be familiar with ontology representation languages, but finds it cumbersome and challenging to describe the required Web Service in it.

- **Normal user**

  This user is typically the user found in the "Semantic Web" example in chapter 3. This user will want to use his or hers natural language for describing what he or she requires of a Web Service.

With these users in mind it is clear that the type of Web Service composition has an influence on how the user communicates his or her needs in order to find a Web Service that fulfills that need. A Web Service composition that will perform a specified task will be constructed by people with knowledge of computers and can take advantage of existing description, modelling or programming languages. A semantic web agent that performs more general problem solving for normal users encounters greater challenges in terms of translating from natural language into machine-understandable languages. This takes us to requirement 3.

R3 – Description types:	The means of representing a description of a required Web Service needs to find the right balancing point between human-understandable and machine-understandable.

6.3 Composition requirements

In order to create a serial composition of Web Services you need to make sure that information from one service can be used as input to the other. This means that you need to know the order of the Web Services, both the sequence of invocation of the Web Services and the sequence of the operations within one Web Service. In addition you need to know how to deal with data that needs to be converted from one form to another.

6.3.1 Operation ordering

A complex Web Service might have several different operations that have to be called in a specific order, and in order to use such services in a composition requires that this order is known. A flight booking Web Service might for instance require that you first specify the places you want to travel between so that it can find suitable flights, and then you can buy a ticket for the flight. If you have to interact with the Web Service in this order, it needs to be specified so that the software using this Web Service actually will interact with the service in this order.

R4 – Operation order:	The order of execution among the operations in a Web Service needs to be defined.

6.3.2 Invocation ordering

When using Web Services in a composition you needs to know what services will provide information that will be used by other services, this will determine the order of invocation of the services. In addition you need to know what services requires input from a human user.

R5 – Service order:	The order of invocation among the Web Services in a composition needs to be defined.

6.3.3 Input/output mappings

Web Services consist of operations that given some input will provide output. By using Web Services in a composition, information provided by one service that is going to be input to another service needs to have a relationship to the expected input of the service using the information, that at least falls into the semantic relevance category presented in chapter 2. In some cases the output can be used directly as input to the other service, but more often you might have to do a transformation of the output so that it matches the expected input. The possibilities and what to do in different situations will be presented here. These possibilities are adapted from [K94], and put in to a Web Service composition context. Since Web Service consist of operations that transform input into output the situation is the same as when comparing attributes in [K94] to asses if they are semantically similar or not. The issues that might arise concerning attributes are described below and summarized in table 5. When determining attribute conflicts, similarities or differences in the names and domains have to be determined first.

This gives us the following possibilities:

- **Same name, same domain.**

  When the names of the two attributes are the same and they have the same domain, the attributes either have the same interpretation, which means that they are semantically equivalent, or they are homonyms, meaning that they are semantically incompatible. If the output from a Web Service operation has the same name and same domain as the expected input of another and they have the same interpretation the output can be used directly as input to the other service.

- **Same name, different domains.**

  When the name is the same but the domains are different, there are four possibilities.

  1. The representations may cause a conflict, i.e. if the value in one attribute is represented as a character string and as an integer in the other.

  2. There might be a scaling issue, i.e. one attribute has values in inches while the other has centimeters.

  For these two types of issues in most cases it is possible to establish a total one-one mapping, making them semantically equivalent. This means that if you can establish a total one-one mapping simple conversions suffice in order to use the output as input.

  3. There might be a data precision issue, i.e. the domain of one attribute is coarser than the domain of the other, in this case there is a many-one mapping from the less coarse domain to the coarser. In this case the attributes are semantically related. This means that if the expected input is the coarser domain a conversion of the output can be used as input. If the expected input is the less coarse domain it will depend on if the Web Service tolerates imprecise input.

  4. The last possibility is that the attributes are homonyms.

- **Different name, same domain.**

  If the attributes are referring to the same concept, and the only difference is in their name, the attribute names are synonyms making the attributes semantically equivalent. This situation lets the output be used directly as input by the other service. If the attributes are not referring to the same concept they are semantically incompatible.

- **Different name, different domains.**

  In this case the possibilities are the same as for attributes with the same name and different domains, except of course, that they can't be homonyms since the don't share the same name, but they still might have a different interpretation and thus be semantically incompatible.

- **Default values.**

  If an attribute has a default value, two possibilities arise. Either the default values are the same, in this case we can ignore the value and treat it as an attribute without a default value, or the values are different. When the default values are different there are no natural mappings between the default values, but the attributes might play the same roles, thus semantically resemble each other. This situation is irrelevant to Web Services as input and output by their nature won't have default values.

- **Integrity constraints.**

  Integrity constraints on attributes are logical expressions that must be satisfied by every value the attribute takes. Only values from the domain of the attribute that satisfies the constraint are legal. If, for two attributes, their constraints are satisfied by different sets of values, we have a conflict. If the constraints on, in other ways comparable, attributes are inconsistent, we may still establish a semantic similarity, since the attributes play the same roles. If the integrity constraints don't exclude each other entirely, stronger similarities than semantic resemblance might be established.

Below follows a table summing up the attribute conflicts that might arise.

Attribute conflicts	Same name, same domain.	Homonyms	Semantic incompatibility
		Same interpretation	Semantic equivalence
	Same name, different domains.	Data representation	Semantic equivalence
		Data scaling	Semantic equivalence
		Data precision	Semantic relevance
		Homonyms	Semantic incompatibility
	Different name, same domain.	Synonyms	Semantic equivalence
		Different interpretation	Semantic incompatibility
	Different name, different domains.	Data representation	Semantic equivalence
		Data scaling	Semantic equivalence
		Data precision	Semantic relevance
		Different interpretation	Semantic incompatibility
	Default values		Semantic resemblance
	Integrity constraints		Semantic resemblance

Table 5- Attribute conflicts

In all the cases above, the most challenging part is determining whether there are any relationships between the attributes (input/output) under consideration or not. If you for instance have two attributes named foo and both domains are all integers, there is a difficulty in determining whether these attributes have the same interpretation or if they are homonyms. The complexness or simplicity of this will be determined by the value of the metadata. This means that the output and input needs to be described in a manner that makes it possible to determine if there are any semantic relationships.

R6 – Semantic descriptions:	Web Services input and output must be described in a way that makes it possible to establish semantic similarity between them.

6.3.4 Alignment with the Semantic Web

The vision of the Semantic Web where web content is machine-processable has produced several important standards aiming at fulfilling the vision. Web Service composition is by many seen as the silver bullet in terms of realizing the Semantic Web [MBE03], and any approaches aiming at creating compositions should therefore follow the vision by making use of the already defined standards. Failing to follow the work by the Semantic Web community will probably result in failing to get enough support for the approach by the Web Service community.

The W3C group working on the Web Service architecture (WSA) has defined a set of requirements for the architecture that ensures that the W3Cs work on semantic interoperability is not set aside [W3CWSA]. The WSA should avoid any unnecessary misalignment with the Semantic Web. In other words, new Web services technologies, developed by W3C Web Services WGs, should be capable of being mapped to RDF/XML. All conceptual elements should be addressable directly via a URI. The WSA must not preclude the characterization of a Web Service that attempts to make its semantics clear to an automatic system using technologies such as those adopted as part of the Semantic Web. Web service descriptions should be capable of referencing concepts identified by a URI in an ontology, such as W3C OWL.

R7 – Semantic Web alignment:	Web Service composition approaches have to be in alignment with the Semantic Web vision.

6.4 Summary of the defined requirements

The defined requirements were split up in to design requirements, discovery requirements and composition requirements. With the exception of requirement 1, all requirements are applicable to the general case of creating a Web Service composition whether it's for a general problem solving agents or for a Web Service or application that is created for a specific purpose. Table 6 shows a summary of the requirements defined above.

R1 – Change tolerance:	Web Service compositions that are set up to perform a specific task repeatedly will have to be tolerant to change in order to achieve interoperability
R2 – Functionality matching:	Web Service discovery needs to be done by matching of functionality.
R3 – Description types:	The means of representing a description of a required Web Service needs to find the right balancing point between human-understandable and machine-understandable.
R4 – Operation order:	The order of execution among the operations in a Web Service needs to be defined.
R5 – Service order:	The order of invocation among the Web Services in a composition needs to be defined.
R6 – Semantic descriptions:	Web Services input and output must be described in a way that makes it possible to establish semantic similarity between them.
R7 – Semantic Web alignment:	Web Service composition approaches have to be in alignment with the Semantic Web vision.

Table 6- Requirements

6.5 Evaluation of existing approaches.

In chapter 4 two approaches to Web Service composition were presented, the standards approach and the DAML-S approach. In this section we will evaluate these approaches in terms of the requirements that were defined in this chapter. The approaches will be evaluated in terms of one requirement at a time. In section 6.5.8 the individual evaluations will be contrasted against each other, and the points of improvement will be explored.

6.5.1 Evaluating the existing approaches in terms of requirement 1

R1 – Change tolerance:	Web Service compositions that are set up to perform a specific task repeatedly will have to be tolerant to change in order to achieve interoperability.

The important thing concerning this requirement is that developers that are developing software that make use of Web Service composition come up with some plan of how to deal with Web Services that are down or changed in any other way. Ultimately, there is no way to deal with every possible change that may happen, sometimes the only reaction possible is to say; sorry, your request cannot be fulfilled. If reasonable plans to react to change, like a list of alternative Web Service, are existent this requirement will be partially fulfilled, but to be fully fulfilled an approach will have to make

use of automatic discovery.

BPEL4WS being an execution language can set up alternative services for a specific task. Choosing among them will be determined by control constructs and fault handling. DAML-S on the other hand, provides only a description of the process, and therefore setting up alternative services is handled outside the scope of DAML-S.

If alternative services are set up the DAML-S process model lets you describe alternative services for a specified task and applying a control construct to choose among them. The use of the control construct choice, doesn't show the reason for the choice as the construct is non-deterministic, and thus other control constructs have to be used to describe the choice.

6.5.2 Evaluating the existing approaches in terms of requirement 2

R2 – Functionality matching:	Web Service discovery needs to be done by matching of functionality.

This requirement underlines the superiority of the DAML-S approach when it comes to Web Service discovery. UDDI does not offer any possibilities of Web Service discovery based on functionality. UDDIs search by categories will not suffice for automated discovery as it says nothing about what the Web Service does.

The DAML-S approach depends on the use of ontology markup which means that the markup has to be of a certain quality to ensure that the semantic descriptions are adequate for automated reasoning. While the work on ontological markup languages is still ongoing and cannot express all the semantics that may be needed, the DAML-S approach has the right angle on how to perform Web Service discovery, both at run-time and at development-time.

6.5.3 Evaluating the existing approaches in terms of requirement 3

R3 – Description types:	The means of representing a description of a required Web Service needs to find the right balancing point between human-understandable and machine-understandable.

Searching for a Web Service in a UDDI registry is easy from the human perspective, browse the categories or search by keywords, however since UDDI fails requirement 2 it fails this requirement too. If you can't discover Web Services based on functionality, it makes no sense to describe the required functionalities, and keywords together with categories lack sufficient semantics to be machine-understandable.

DAML-S on the other hand suffers from making use of a rather complex markup language which

takes a while to learn. In [SRS] the authors have done an experience report on DAML-S and one of their main worries is the difficulty in getting started with writing DAML-S. This concurs with our experiences as well, it requires experienced users to describe a Web Service in DAML-S.

6.5.4 Evaluating the existing approaches in terms of requirement 4

R4 – Operation order:	The order of execution among the operations in a Web Service needs to be defined.

Both the DAML-S approach and the standards approach lets you describe the Web Services as processes and thus provides enough information for determining any specific ordering of the operations that has to be followed. DAML-S lets you define composite processes that can handle this requirement and BPEL4WS can define correlations grouping related Web Service operations and ordering among them in addition to control constructs.

6.5.5 Evaluating the existing approaches in terms of requirement 5

R5 – Service order:	The order of invocation among the Web Services in a composition needs to be defined.

Both the DAML-S approach and the standards approach lets you define the order of invocation among the Web Services by using control constructs defining the process.

6.5.6 Evaluating the existing approaches in terms of requirement 6

R6 – Semantic descriptions:	Web Services input and output must be described in a way that makes it possible to establish semantic similarity between them.

The standards approach relies on WSDL to describe the input and output of operations that belongs to a Web Service. These descriptions contain little or no semantics, only datatypes and possibly restrictions on them, making it difficult to establish any semantic similarities. This puts a responsibility on developers to use names on operations, input and output that are intuitive, however this situation only helps humans using the descriptions, computers have no way of understanding the semantics of a name, either the name is the same or it's not the same, if it means the same or not will remain unknown.

DAML-S lets you describe input and output using ontological markup, which makes it possible for a computer to determine if there is a semantic similarity. DAML-S says nothing about how you define the ontologies, and interpreting from the examples provided by the DAML Services Coalition a multiple ontology approach is used. This, as explained in chapter 3, faces us with the problem of interontology matching. A clearly defined strategy for use of ontologies would be very helpful.

6.5.7 Evaluating the existing approaches in terms of requirement 7

R7 – Semantic Web alignment:	Web Service composition approaches have to be in alignment with the Semantic Web vision.

For the standards approach we have a partial alignment, with the standards being based on XML, but there lacks means of using OWL descriptions. However there are initiatives for aligning BPEL4WS with the semantic Web [MM03]. When it comes to WSDL, the datatypes there should also be possible to represent using references to concepts in ontologies.

The DAML-S approach is in complete alignment with the "Semantic Web", especially when it evolves into OWL-S, which is one of the main technologies of the "Semantic Web" vision.

6.5.8 Summary of evaluation

Table 7 sums up the evaluation of the approaches to Web Service composition by comparing the two approaches in terms of if they fulfill the specific requirements or not. A (+) sign represents a requirement that is fulfilled, a (–) sign represents a requirement that is not fulfilled. (+/-) represents a requirement that is only partially fulfilled.

	WSDL/UDDI	**DAML-S**
R1 – Change tolerance:	+/-	-
R2 – Functionality matching:	-	+
R3 – Description types:	-	-
R4 – Operation order:	+	+
R5 – Service order:	+	+
R6 – Semantic descriptions:	-	+/-
R7 – Semantic Web alignment:	+/-	+

Table 7 - WS composition approaches in terms of requirements

As we can see the DAML-S approach is superior with respect to developing semantically interoperable Web Service compositions. DAML-S strong relation to ontologies in effect of itself being an ontology is the main factor of the victory over the standards approach in this evaluation. In addition DAML-S also enables functionality based Web Service discovery, whereas the standards approach relies on keywords. The only point where both approaches does not fulfill the requirement is when it comes to having a balance between human understandable and machine understandable descriptions. This situation we will try to improve with our proposal, MOSIS, in chapter 7. In addition we will introduce improvements on the requirements that are not fully fulfilled by the two approaches.

6.6 Convergence of DAML-S and UDDI/WSDL.

In light of our analysis of the Web Service composition approaches, the DAML-S approach seems to be the best suited approach. However there are other forces that are in the way of wide spread use of DAML-S. The three main Web Service standards, SOAP, WSDL and UDDI have become synonymous with Web Services and they have been embraced by many of the important parties in the computer industry, like Microsoft and IBM. Such marketing forces cannot be ignored even if the DAML-S approach is seemingly better for Web Service compositions. As a response to this several of the actors involved with DAML-S are working out solutions to the convergence of DAML-S and these three important standards.

- **WSDL and DAML-S**

  DAML-S concept of grounding is generally consistent with WSDL's concept of binding [DAMLS]. Using DAML-S and WSDL together can be done by using the extensibility elements already defined in WSDL, the only exception to this is an extensibility element, proposed in [DAMLS], that relates an DAML-S atomic process to a WSDL operation. Figure 16 shows the mapping between DAML-S and WSDL, where DAML-S atomic processes correspond to WSDL operations, and a WSDL message can be described using a description logic based type such as a DAML+OIL or OWL type.

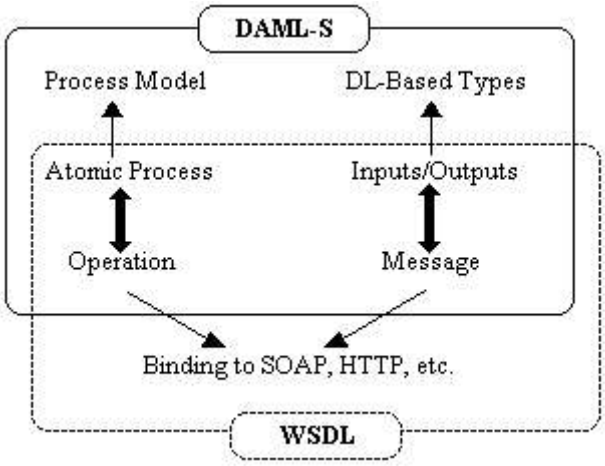

Figure 16 - Mapping between DAML-S and WSDL [DAMLS]

DAML-S has a subclass of grounding in which the relevant WSDL constructs can be referenced in DAML-S called WSDLGrounding.

- **SOAP and DAML-S**

  As an effect of the relationship between DAML-S and WSDL presented above, there are no problems involved when using DAML-S and SOAP together. Thus, with the extension proposed in [DAMLS] to relate atomic processes to operations, grounding DAML-S with WSDL and SOAP is possible.

- **UDDI and DAML-S**

  In order to use a UDDI registry for referencing to Web Service with a DAML-S description, UDDI will have to support a functionality based matching. [PKPS02] suggests an approach to doing this by augmenting UDDI registries with an additional semantic layer that performs the capability based matching. This approach facilitates Web Service discovery using either UDDI keyword search or DAML-S functionality matching.

As we can see the convergence of DAML-S and the three standards is not an impossible task. Relatively minor adjustments will make it possible to exploit the advantages of DAML-S while at the same time exploiting the industry backing of the three standards.

6.7 Summary

This chapter defined seven requirements for Web Service composition, covering design aspects, the discovery phase and the composition phase. These requirements were used to evaluate two approaches to Web Service composition, and the DAML-S approach came out as the winner of the evaluation because of its use of ontologies and its strategy for Web Service discovery. Finally we presented some ideas on how to combine the two approaches.

7. MOSIS

Composing Web Services is a difficult task as it involves being able to assess the semantics of the Web Services in the composition. In order to discover Web Services you need to understand what they do and in order to chain the services together you need to understand the meaning of the input they need and the output they provide so that you can develop mappings from output to input. As we have seen in chapter 6, the two main approaches for composing Web Service only partially fulfill the requirements for Web Service composition. In this chapter we will present our approach for dealing with Web Service composition which covers both the discovery phase and the composition phase. We have called our approach MOSIS (**MO**delling **S**emantic **I**nteroperable web **S**ervice compositions). MOSIS focus on the use of models along with concepts from MDA, and the use of ontologies. Following the MOSIS approach will mean using a development tool capable of UML modelling, generating DAML-S descriptions and accessing ontology repositories (fig. 17). This means that MOSIS is similar to the DAML-S approach, only extending it with models to enhance human interpretation of Web Service descriptions. Acknowledging the importance of the standard UDDI, MOSIS assumes that UDDI registries with an added semantical layer are used as in [PKPS02]. We want to emphasize that MOSIS is aimed towards what we called specific purpose systems in chapter 3. MOSIS can be seen as a recipe to follow when developing Web Service composition and as a layer on top of DAML-S providing a different view.

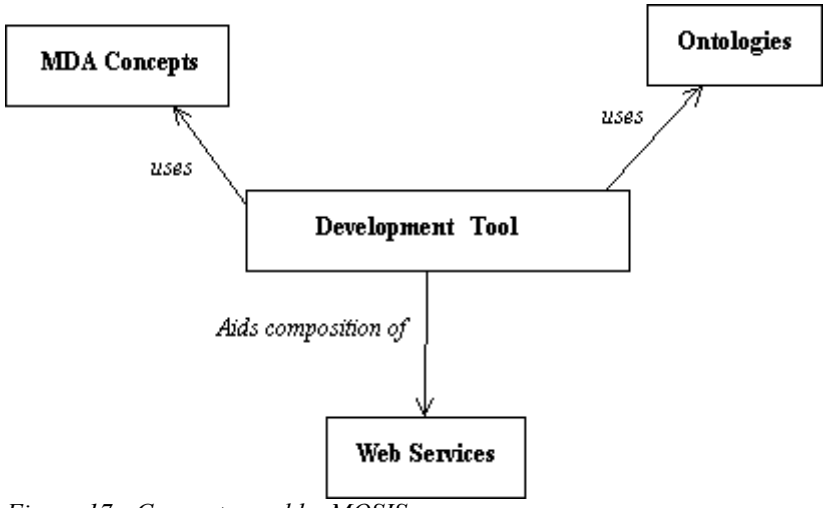

Figure 17 - Concepts used by MOSIS

First we'll present an overview of MOSIS and then go into the more specific details of the elements directly concerning semantic interoperability. These elements are, modelling for discovery, modelling for composition, and modelling ontologies. With regards to these elements, MOSIS relies on the ACE-GIS project presented in chapter 5. MOSIS brings the work of [ACEGIS2] (modelling for composition) and [ACEGIS3] (ontology architecture) together, while adding modelling for discovery and a UML approach to ontologies. Finally we'll give some details concerning how MOSIS can be realized.

7.1 Overview of MOSIS

MOSIS covers both the discovery and the composition phase of Web Service composition. We want to use UML models in the discovery of Web Services in order to put it into a more human friendly context than DAML-S descriptions and ontology representation languages does. The way we want to achieve this is by developing the compositions in UML development tools that support UML and have them generate the necessary DAML-S descriptions and/or ontology representation language descriptions. For the discovery phase the development tool should use a UDDI with an added semantical layer to find services. The added semantical layer deals with the functionality matching of the description of a requested service and the descriptions of the Web Services in the registry. In this aspect we have adopted the approach of the DAML-S/UDDI Matchmaker presented in [PKPS02]. Figure 18 illustrates the architecture of the DAML-S/UDDI Matchmaker and how our approach with a development tool will interact with the Matchmaker.

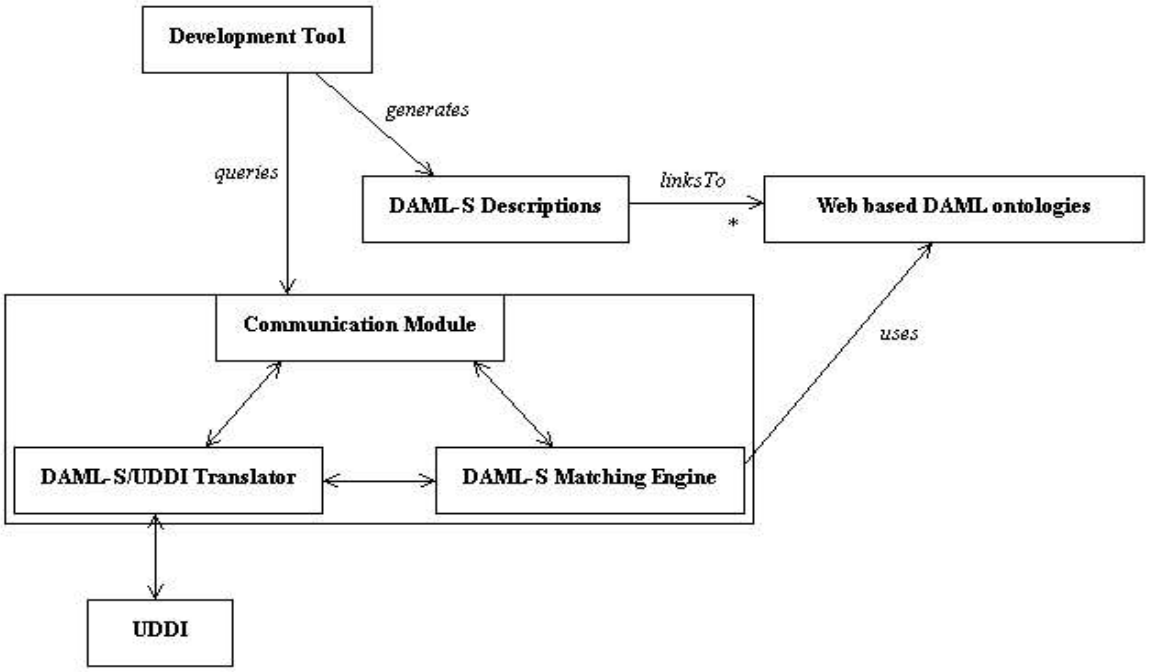

Figure 18 - The architecture of the DAML-S/UDDI Matchmaker from [PKPS02] in interaction with a development tool.

The development tool lets the developer use UML to model the discovery process, where all requested Web Services are identified, the information they use is linked to web based ontologies and the DAML-S code and possible ontology descriptions that does not exist are generated. Then the tool lets the developer use UML to describe the actual Web Services that will be used along with the possible mappings that tie them together. Figure 19 shows the composition process with the tasks that need to be performed.

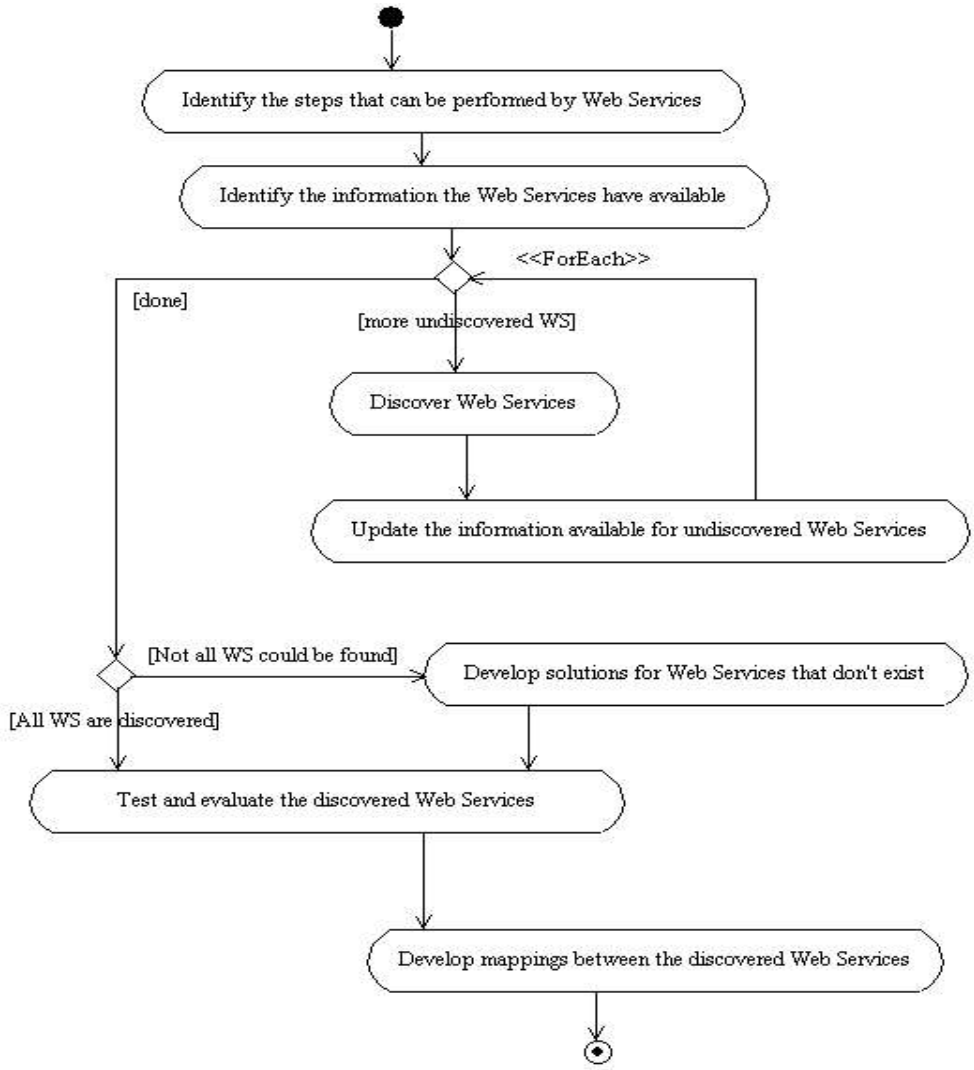

Figure 19 - The Web Service composition process

The steps in figure 19 indicates the recipe for how Web Service compositions are developed using MOSIS. Each of the steps are associated with the development of models in UML. The Web Service composition process uses four different models. The discovery model, the composition model, the information discovery model and the information composition model. Following the MOSIS approach means that the development of the MOSIS models are done in the step by step fashion of figure 19.

The information models model the Web Services during two phases, the information discovery model models the Web Services we want to discover, and the information composition model models the Web Services we have found as a result of the discovery phase. These models are realized as UML class diagrams, where Web Services are classes with operations. Associated with the Web Services is the information they consume and produce. In the information discovery model, this information represents the output you request from a Web Service you want to discover and the input you have available for the service to produce that output. In the information composition

model, the information is the real input and output the services you have discovered consume and produce. The information is realized in UML class diagrams, where the information are classes with attributes and no operations. To enable semantic matching the information will have ontology references. The information composition model thus gives us structure which can be realized as XSD datatypes with ontology references in a WSDL document which is realized by the information composition models services. In the information discovery model, the ontologies are the most important aspect as they will be used for functionality matching, additional information is used as selection criteria.

To define the process that a Web Service composition realizes we use UML activity diagrams. The discovery model shows the ordering and information flow by referencing to the requested Web Services and their associated information in the information discovery model. The composition model shows the ordering, information flow and data mappings by referencing to the real Web Services that were discovered in the discovery phase. Table 8 shows the realization of the MOSIS models.

	UML	***Realization***
Structure	Class diagrams representing the information the Web Services consume and produce in the information models.	XSD with ontology references.
Service	Class diagrams representing the Web Services in the information models.	Composition: WSDL documents representing the Web Services. Discovery: DAML-S profile.
Process	Activity diagrams in the discovery model and the composition model.	Composition: DAML-S process model representing the process realized by the composition.

Table 8 - Realization of the four MOSIS models

Below we explain the steps in the Web Service composition process, indicating where the different models are developed.

- **Identify the tasks that can be performed by Web Services**

  The goals of a project using a Web Service composition can by achieved by performing certain tasks. Some of these tasks have a characteristic that makes it suitable for being performed by a Web Service. Being suitable for being performed by a Web Service means that it is likely that there already exists a Web Service performing that task. Upon identifying these tasks, the way that they will achieve the goal of the project will depend on the order the of the tasks which will also have been identified by the investigation on how to reach the goals. Linking subprocesses achieving subgoals together will define a process that achieves the main goals, thus we can use a UML activity diagram to aid the discovery process. Since the Web Services we find in the discovery process will depend on the information we can provide them and the information they provide, having a clear picture of how the Web Services will be used to reach goals will greatly

aid the discovery. This step defines the order of the requested services by defining them in the information discovery model and indicating the order in the discovery model.

- **Identify the information the Web Services have available**

  With the specification of the order of the tasks to be performed we have an indication of the information flow through the composition, this should be used to define the information we have and need in a way that makes the semantics clear enough to be used in functionality matching. Specifically this means that every piece of information should be linked to already existing ontologies. In addition to this a specific ontology for each of the Web Services should be developed. The reasons for this will be explained in section 7.3.2. The information is defined in the information discovery model and will by referenced from the discovery model indicating the information flow.

- **Discover Web Services**

  The information flow determined by the order of the tasks to be performed by Web Services along with links from the information to ontologies makes it possible to create DAML-S profile descriptions of the requested Web Services. The DAML-S profile is then matched against descriptions of existing Web Services. The discovery model and the information discovery model created by the two preceding steps will be used to generate the DAML-S profile descriptions.

  All matching Web Services should be kept along with services that only partially fit the description of the requested service. These should be kept as alternative services in case the primary selected service becomes unavailable at some point. This means that services that only partially fit the descriptions for the requested services will have to at least provide the requested output. Misalignments in input can in some cases be corrected by providing additional information from other sources. The set of input information that is available does not need to be fully matched, it suffices that at least one element is required by a Web Service.

  This step has to be repeated until all Web Services are discovered. This repetitiveness forces an incrementally developed discovery model and information discovery model as both new information and new tasks might be introduced.

- **Update the information available for undiscovered Web Services.**

  When one or more Web Services have been discovered, the remaining undiscovered Web Services might have more information available as a result of discovered services that provide more than just the requested output. This means that more Web Services for a specific task might be discovered. You want to widen your search for Web Services as much as possible to ensure that you find the best suited service possible. This step will update the information discovery model.

- **Develop solutions for Web Services that can't be found.**

  Not all tasks you want performed by a Web Service will have an existing Web Service that you can use. In this case you will have to find another solution, for instance developing the missing Web Service yourself.

- **Test and evaluate the discovered Web Services.**

  All discovered Web Services will have to be tested and evaluated in order to make sure that they perform the tasks the description say they will do. Below we have enlisted some criteria that may be useful for evaluating the discovered Web Services.

  Evaluation criteria:
  - **Accuracy** – Does the Web Service actually provide results that you can use.
  - **Security** – Does the Web Service satisfy your required level of security.
  - **Time** – Does the Web Service provide the result in a time frame that is acceptable.
  - **Cost** – Is the price for using the Web Service acceptable.

- **Develop mappings between the discovered Web Services.**

  This method of Web Service composition ensures that the information flowing through serial compositions will match semantically. However there might still be syntactical and structural differences that will have to be dealt with. We have chosen to deal with this the same way as in the ACE-GIS project [ACEGIS2], where they also make use of UML activity diagrams to model the composition. This diagram is used to generate workflow XML that makes it possible for a workflow engine to execute the composition. The composition model only differs from the discovery model in two aspects. The Web Services in the composition model are real Web Services that have been discovered, not only services that you want to discover. The composition model also incorporates the mappings needed to make output from one service useful as input to another service. The information composition model defines the Web Services in the composition and the information they consume and produce. The information should have references to ontologies developed by the developers of the respective Web Services.

7.2 MOSIS and MDA

There are three reasons to why we want to incorporate MDA concepts in MOSIS. First of all, the focus on architecture in MDA in terms of how different systems fit together is something which is the main focus of MOSIS as well. MDAs use of models to increase the understanding of how, when and why different systems interoperate has been influential to why we have decided to use models in MOSIS.

Secondly, the notion of being able to generate code based on models in MDA is reflected in MOSIS where ontology descriptions will be generated from models. Although the focus in MDA has been on generating J2EE and .NET code, generating DAML-S, DAML+OIL and OWL descriptions can greatly improve both the readability and the ability to create these descriptions by others than

ontology representation language experts.

The third reason comes as a consequence of code generation, as keeping ontologies as UML models and generating the descriptions makes it possible to have the same ontologies in different ontology representation languages. This also makes it possible to keep the knowledge even if new and improved ontology representation languages emerge. You will only have to create new code generators to translate all of your models of ontologies.

7.3 MOSIS and ontologies

The process indicating the order of the Web Services in a composition will be used to identify what information is expected to be available. The next step then is to add semantics to the information, so that it can be used in the Web Service discovery process. Clear semantics will also aid the process of defining mappings between services in the composition phase. This section will look at how ontologies can be represented by UML, and will then introduce an ontology architecture that will provide a strategy for using ontologies in MOSIS.

7.3.1 Ontologies and UML

One of the issues concerning ontologies is the difficulty in developing them. The ontology representation languages generally lack tool support and developers fluent in them are rather few. This has brought on discussions on how to make ontology development easy enough to be done by the large number of domain experts that exist for pretty much all domains possible. The true experts in a domain are the ones dealing with issues concerning the domain on a regular basis, not the ones fluent in ontology representation languages. If ontology development became an easy task to pursue, a larger number of people would get involved in the making of ontologies and more and more of them would emerge. The main part of the problem is that ontology representation languages are created to be machine-interpretable. In addition to this there is a very limited number of tools for ontologies with the possibility of representing the ontologies graphically. Fortunately, there is another technology that aims at modelling the world with substantial support in both human-knowledge and tools, but from the perspective of being human-interpretable. In this aspect ontology representation languages and UML are complementary, but there are some conceptual complications that must be overcome to provide a 1-1 mapping between them. Establishing a relationship between UML and an ontology representation language can take two directions, either UML is used as the graphical notation for ontology representation languages or already specified knowledge in UML is made publishable on the web by transformations to an ontology representation language. The difference in viewpoints results in two different questions about the transformation. How to map ontology representation languages into UML, and how to map UML into ontology representation languages.

- **Mapping ontology representation languages to UML**

  Mapping ontology representation languages into UML with the objective of using UML as a graphical notation for ontology representation languages presents a difficulty in representing the notion of property. Property is part of many ontology representation languages such as DAML, DAML+OIL and OWL . A property corresponds to a UML association, but there are some differences; a property can exist as an independent element without being attached to other constructs. In contrast UML associations are connected to classes with association ends, thus making each association unique. This means that UML associations can only express local restrictions on classes, while for instance DAML properties along with other elements can express local as well as global restrictions [FSS03]. Baclawski has proposed a solution to this by extending the UML meta-model. He suggested to enrich the Meta-Object Facility (MOF) specification with the notions of *property* and *restriction* as it is shown in figure 20.

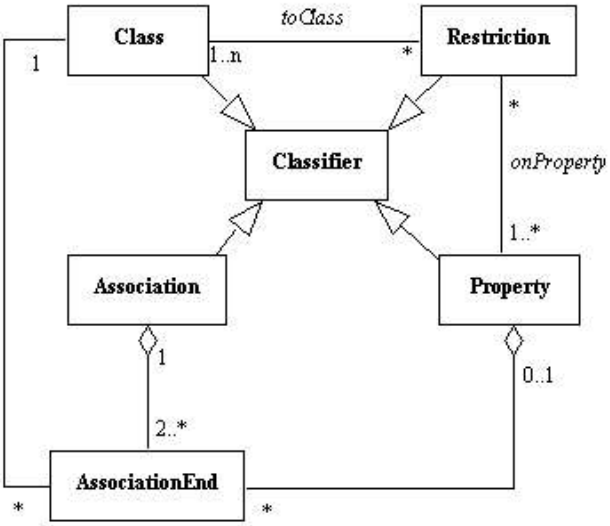

Figure 20 - UML metamodel extension for property restrictions [FSS03]

- **Mapping UML to ontology representation languages**

  Mapping UML to ontology representation languages with the intent of capturing already specified knowledge means representing all UML elements in a ontology representation language. Falkovych has proposed a transformation of UML diagrams into DAML+OIL ontologies that preserves the semantics of UML concepts [FSS03]. In this work mapping associations to properties can be done by mapping to *daml:ObjectProperty* using a unique identifier for the property name, thus making it possible to distinguish different associations with the same name as different DAML+OIL properties. UML attributes can also be mapped in the same way since they also have a local scope within its class.

Both of the different transformation viewpoints presented above are applicable in the Web Service composition context. For someone developing a Web Service composition, ideally you want to use

already existing ontologies and these might only exist in ontology representation languages and by mapping to UML they might be easier to understand and use. In cases where you might have to develop the ontologies yourself, doing it in UML saves you the trouble of having to hire an expert in ontology representation languages.

7.3.2 Using ontologies in MOSIS

The use of ontologies in MOSIS occurs at two points. First of all we need to define the semantics of the IOPEs (Input, Output, Precondition and Effect) used in our DAML-S service profile so that it can be used for matching against profiles of existing services in the discovery phase. This of course assumes that the IOPEs in the profiles of the existing services also is described by ontologies, something they by definition are. Secondly, the IOPEs of the Web Services that are chosen to participate in the composition needs to be described by ontologies, so that the semantics can be used to develop mappings between the services in the composition. The way we want to use ontologies is the same as proposed used in the ACE-GIS project [ACEGIS3], where the authors introduce a three level ontology architecture.

The three level ontology architecture consist of an application level, a conceptual level and a semantic grounding level. Figure 21 shows the relationship between these levels.

- **Application level**

  On this level an application ontology is used to capture the world view of the services. The names of the elements in a WSDL document, such as messages and operations, are related to each other with non-taxonomic relations. The idea is to represent how the service models the information it consumes and provides. Concepts from the application ontology are referenced to concepts in a domain ontology, which exist on the conceptual level. Application concept references can put restrictions on the properties of the domain ontology concept, constraining the meaning of the domain ontology concept. Concepts defined in an application ontology are expected to be dynamic symbols subjected to frequent changes. The number of application ontologies is expected to be high as every Web Service or almost every Web Service will have an application ontology of its own.

- **Conceptual level**

  On this level domain ontologies are used to describe a domain from a certain perspective. Such a domain can for example be the meteorology domain or the chemistry domain, to name two from the MODUSA case example in chapter 5. These domains will serve to represent the vocabulary used by the actors residing in the respective domains. Concepts in a domain ontology may have more meanings in different domains or contexts, but reflects the domains they are used in. On this level no data types or data models are used, the concepts on this level reflects human concepts, i.e. concepts that are meaningful to humans familiar with the domain in question. The references and restrictions from application ontologies to domain ontologies are used to supply meaning to the concepts in the application ontologies. To provide further grounding to the human concepts in the domain ontologies, the concepts in domain ontologies have references to

concepts on the semantic grounding level. The domain ontologies are intended to provide a stable meaning that are subject to changes only rarely. The number of domain ontologies is expected to be relatively small compared to application ontologies. The number will reflect the number of different domains which are useful in IT projects involving interoperability issues.

- **Semantic grounding level**

The grounding level reflects ideas that are currently only on the research level. The basic idea behind the grounding level is to escape the vicious cycle of continually defining concepts using concepts on an ever higher level of abstraction. Domain ontologies relate human concepts to each other and as a consequence of this the number of ways the concepts can be interpreted will be restricted, but it still cannot be assumed that everybody will have the same understanding of a certain concept. Therefore the authors of [ACEGIS3] propose to use image schemata to semantically ground concepts in domain ontologies. *"Image schemas fall between abstract propositional structures (such as predicates) and concrete images (such as the spatial mental images in (Barkowsky 2001))"* [ACEGIS3]. Image schemas are mental patterns that recurrently provide structured understanding of various bodily experiences, i.e. a way of representing elements of knowledge into patterns or schemas. They can be seen as generic and abstract structures that help people establish a connection between different bodily experiences that have the same recurring structure. As a consequence of this Johnson and Gärdenfors have stated that meaning involves image schematic structures [ACEGIS3]. The semantic grounding will provide a semantic datum that never changes.

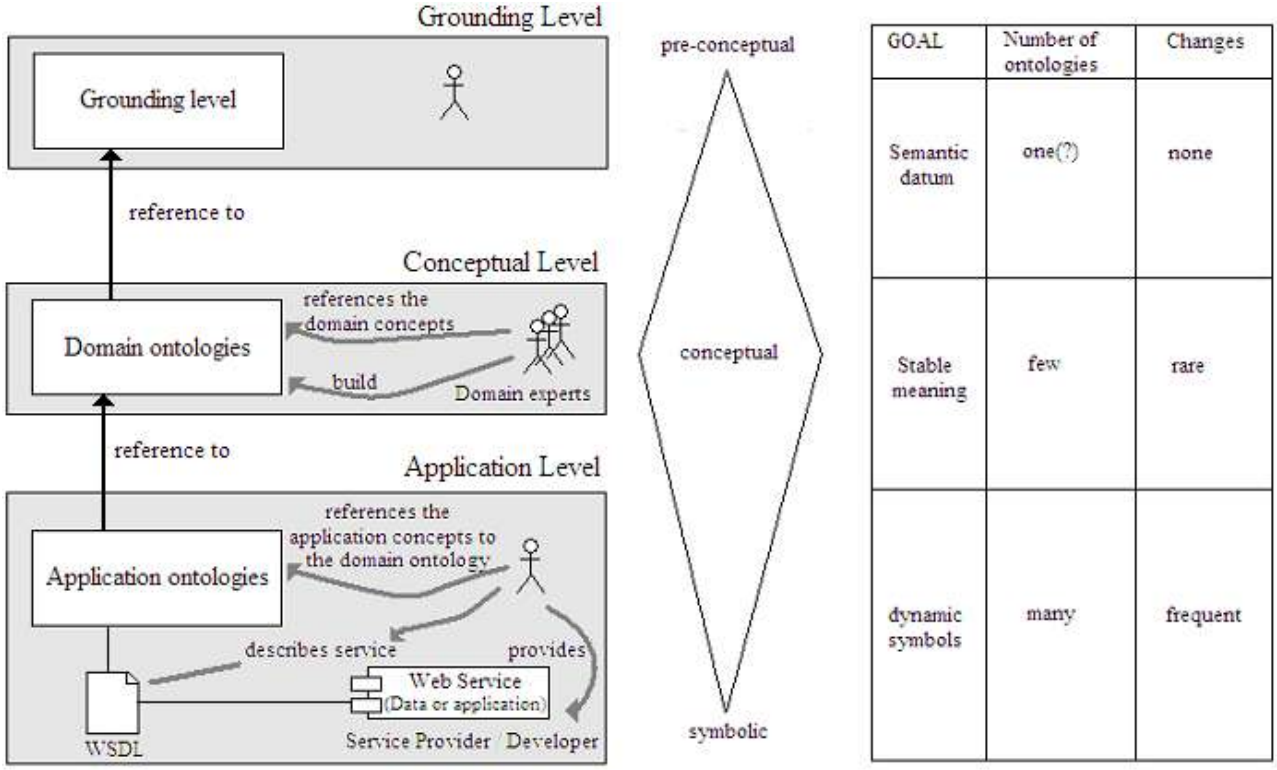

Figure 21 - Three level ontology architecture, adapted from [ACEGIS3]

For MOSIS we can for the time being only use the conceptual level and the application level as the semantic grounding level is still only on the theory plane and will probably stay there for the nearest future. As a consequence of this we will only explain the use of domain ontologies and application ontologies, and assume the existence of a semantic grounding. For the discovery phase ontologies will aid in finding the right services and for the composition phase ontologies will aid in the derivation of mappings between services.

- **Ontologies and the discovery phase**

  In the discovery phase we might not have all the information necessary to create a complete application ontology for the requested Web Services. Nevertheless we should use the information we have to create as complete as possible application ontologies, but still the most important aspect will be to reference the elements to domain ontologies. In order to discover Web Services domain ontologies will be essential for the matching of requested services and existing services. If the domain ontology is the same for both the requested service and an existing service, which it should be as ideally there will only exist one domain ontology for a domain, matching will be straightforward. However there might still be unresolved interoperability issues even if there are references to the same domain ontology. For example, if both the requested and an existing service have references to wind direction in a domain ontology, there might still be differences, such as one represents the wind direction from a point and the other represents the wind direction to a point. Resolving this issue would probably be done on the semantic grounding level. Since the domain ontologies don't include data types or data models, what we know about these should be described in the application ontologies. All the information we have available will be useful for the discovery phase. As a consequence of this, application ontologies for the requested Web Services should be developed with references to domain ontologies, thus providing enough information to match requested services with existing services.

  With respect to the MOSIS models, the references to ontologies will be from the information discovery model, where first the information is defined in a conventional manner in a UML class diagram. This class diagram will have a reference to the application ontology diagram, which is a UML class diagram defining the information in terms of datatypes and restrictions which can be used to generate the application ontology in an ontology representation language. The application ontology will have a reference to a domain ontology which already exists. Figure 22 shows the information discovery model with ontology references.

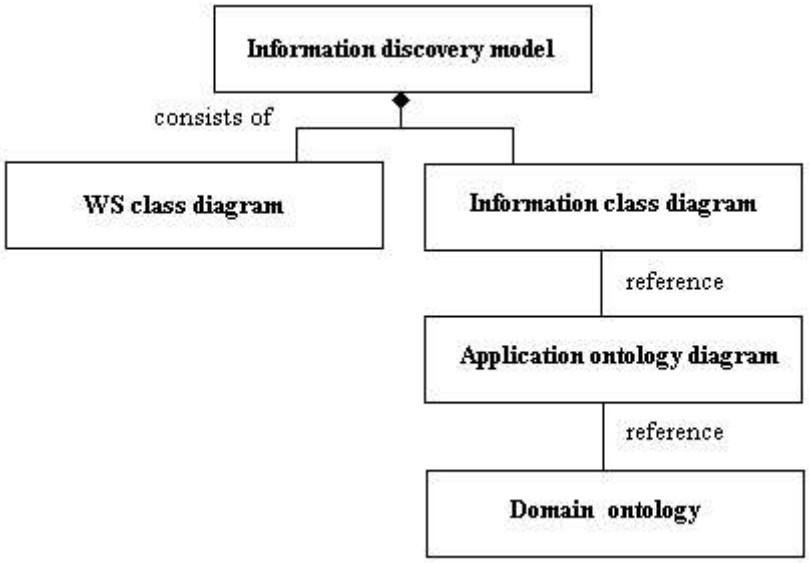

Figure 22 - The information discovery model with ontology references

- **Ontologies and the composition phase**

  In the composition phase we will have access to all the application ontologies involved, assuming of course that the three level ontology architecture is followed by all parties. Even if the architecture is not followed by all parties, the minimum requirement will be that ontologies are used, forcing the problem to be a interontology matching issue. Assuming, however that the Web Services in the composition have application ontologies, we have a good starting position for developing the mappings between the services. The discovery process will have narrowed the existing services down to those who perform the expected tasks and consumes and provides information that is semantically related to the information from eventual other services and the information needed by eventual other services. Knowing the semantic relations lets the data types and data models and other information captured in application ontologies provide a basis for developing the mappings needed. In the same way as for the information discovery model, the information composition model will consist of the actual Web Services and the information they consume and produce with references to application ontologies and domain ontologies.

The three levels in the ontology architecture delegates the responsibility of defining the ontologies to the parties that have the knowledge to do this. Domain ontologies will be developed and maintained by domain experts. Since these ontologies shouldn't exist in several versions for each domain, we envision a repository of ontologies. These repositories should make it easy for developers to find the right domain ontology so they can provide references from application ontologies. Having these ontologies in a form interpretable for developers, for instance UML, and a form interpretable for machines, such as ontology representation languages, will greatly improve the use of ontologies. Developers that are not familiar with ontology representation languages will be able to interpret the knowledge contained in the ontologies. In addition developers will be responsible for developing the application ontologies, and doing this in UML will make sure that knowledge the developers already have can be used.

Following the MDA principles of first describing something independently of platform, describing the information in UML allows for technology changes with regards to the ontology representation language used. DAML+OIL is already becoming outdated as OWL is becoming more and more complete. Capturing the semantics of the information in a technology independent form ensures that technology changes will not make it necessary to go over a new round of defining the semantics in the future.

7.4 The Web Service discovery model.

Determining the order of the Web Services in a composition is very important in order to know what Web Services have what information available. The task of determining the order is similar to defining a business process, where all the steps in the process are specified according to order, conditions and so on. Business processes can be modeled using UML activity diagrams, and this section will show how MOSIS uses activity diagrams to aid in the discovery of Web Services.

We want to use the UML activity diagram as a basis for generating the DAML-S code necessary to perform a functionality based Web Service discovery. In order to generate the DAML-S code we need, we need to know what information each of the Web Services have available. Doing this means associating every step in the process with the information it can make use of and the information it is expected to produce. The steps in the process are the Web Service calls we want to execute and who will use and produce the information. To signify that the steps are Web Service calls we use the *<<WebServiceCall>>* stereotype defined in [ACEGIS2].

Table 9 gives a description of the elements used in the Web Service discovery model.

<<WebServiceCall>>	An action state, which signifies that the step in the process is a call to a Web Service. Corresponds to a Web Service defined in the information discovery model.
Object flow	Represents the information flowing between the Web Service calls. Corresponds to information defined in the information discovery model.
<<Toolstep>>	A step that is initiated by a human actor using a tool.

Table 9- The elements used in the discovery diagram

Figure 23 shows the elements used in the discovery model, where the information produced by Web Service A is used as input to Web Service B.

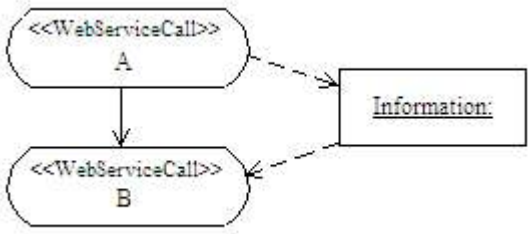

Figure 23 - The elements used in the discovery model

The discovery model illustrating the order of the Web Services in a composition will not necessarily be a static model showing what information is available for a Web Service at all times. Upon the successful discovery of a Web Service the process in the discovery model might have to be updated by indicating the actual information that is available instead of the information that is expected to be available. For instance, if a requested Web Service is one that finds a persons birth date based on the persons name and the birth date will be used as input to another service, you might discover a Web Service that;

1. Produces something that makes it possible to calculate the birth date.
2. Produces the birth date along with other information.

In both these cases the available information for the next Web Services in the composition to be discovered might have to be updated. If its possible that the extra information will be needed in the other Web Services the information discovery model will have to be updated, if it's information that can be considered to be of no interest to the tasks of the other Web Services, updating is not necessary. The Web Service that provides most information in addition to the requested information should be selected as the basis for the updating. This will ensure that the discovery is as wide as possible, finding as many matching services as possible.

7.5 Mapping from UML to the DAML-S profile model.

With a defined order of the Web Services in the composition modeled in the discovery model the expected IOPEs are defined with respect to what information is available at what step in the process. The semantics of the expected IOPEs are defined using links to ontologies. The next step then is to create a DAML-S profile of the requested Web Services so that it can be used to provide functionality matching in the discovery process. The discovery process will be a step by step process where the Web Services are found one at a time. In an incremental manner the information discovery model should be refined on the basis of the Web Services that are discovered. When one or more service have been found that fulfill a specific task, the profile for the next requested service will be generated and a new discovery process initiated.

The mapping from UML to DAML-S profile using the discovery model and the information discovery model is straightforward. The UML object flow going into a UML action state

corresponds to DAML-S *input*, the object flow going out of action state corresponds to DAML-S output. DAML-S *precondition* corresponds to OCL preconditions defined for a <<*WebServiceCall*>> task, and DAML-S *effect* corresponds to OCL postconditions defined for a <<*WebServiceCall*>> task. The actual transformation rules are as yet not defined and implemented.

All of the DAML-S IOPEs need to have a reference to a concept in an ontology, and these should be made explicitly in the classes in the information discovery model representing the information flowing between the services, and in OCL statements connected to the tasks they belong to. The ontology concepts will be from the application ontology and have references to corresponding domain ontologies. Table 10 shows the UML to DAML-S mapping.

UML	***DAML-S***
Object flow going into an action state or preceeding action state.	input
Object flow going out of an action state	output
OCL precondition statements	precondition
OCL postcondition statements	effect

Table 10 - UML to DAML-S

7.6 The Web Service composition model.

In [ACEGIS2] an activity diagram is used as basis for generating INESC workflow XML descriptions. These descriptions can be interpreted by INESC's workflow engine which produces code that automatically handles the Web Service calls and the control and data flow. We have chosen to adopt this strategy in MOSIS, by having a Web Service composition model in addition to the discovery model.

The differences between the composition model and the discovery model are not that great. The composition model will refer to actual Web Services and therefore each step in this process might be a set of alternative Web Services that can be called in case the primary service fail to respond. In addition the information flowing between the services will be the actual information and therefore we need to incorporate the mappings that are needed between input and output in the diagram.

Elements used in the Web Service composition model.

<<WebServiceCall>>	An action state, which signifies that the step in the process is a call to a Web Service.
Object flow	Represents the information flowing between the Web Service calls.
<<MultiServiceProviders>>	Indicates a task that might be fulfilled by a set of alternative Web Services. Contains sub-actions of type <<WebServiceCall>>.
<<DataMapper>>	An action state that signifies that the information flowing into it will be transformed.
<<transformation>>	A note attached to a <<DataMapper>> which gives details about the transformation.
<<SubWorkFlow>>	Indicates that the internal workflow is a workflow of its own.
<<Toolstep>>	A step that is initiated by a human actor using a tool.

Table 11 - The elements used in the composition model

In the same manner as for the discovery model, the composition model is linked to the information composition model where the Web Services and the information they produce and consume is defined.

Figure 24 shows the elements used in the composition model. The information from Web Service A is transformed by the data mapper B and the used as input to the subworkflow C. Finally task D is performed, which is either a call to Web Service E or Web Service F. The Web Services A, B, C, E and F are described in the information composition model along with the information labeled info1 and info2 in the figure.

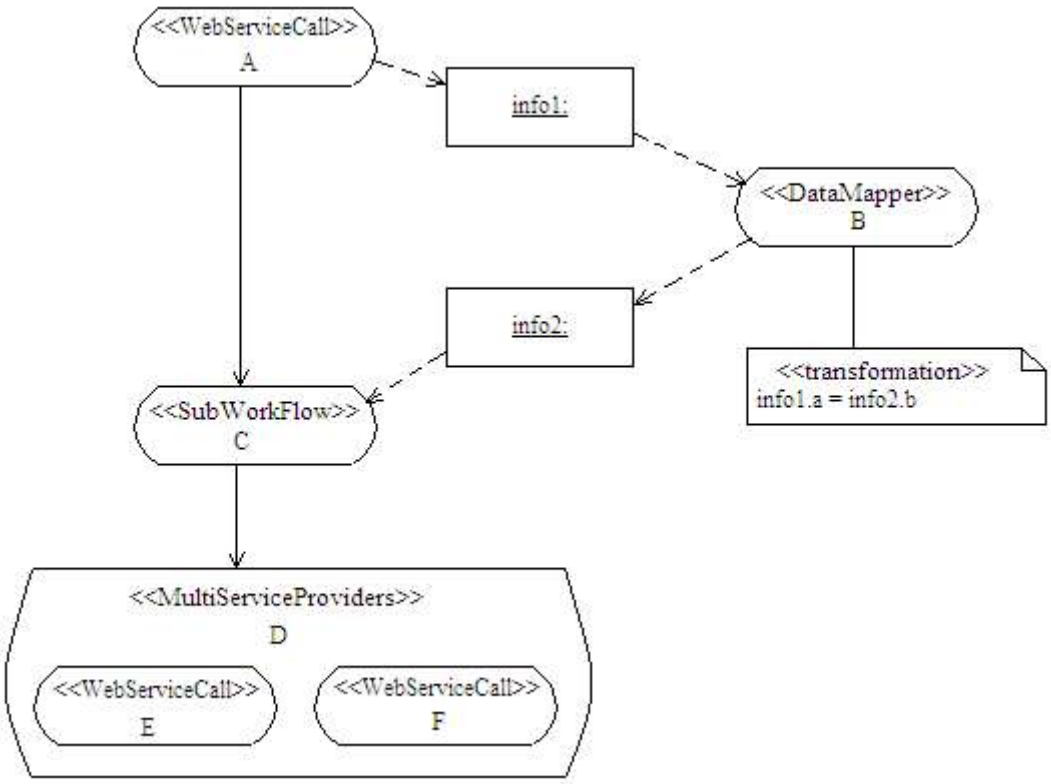

Figure 24 - The elements used in the composition model

7.7 Realizing MOSIS

In order to realize MOSIS we have defined a minimum requirement, which will extend existing UML development tools. In addition we envision a more complete realization based on a platform independent model of the composition contained in a repository, which allows for the different views of MOSIS and can be used by different tools.

7.7.1 Minimum requirement

The realization of MOSIS does not require a full implementation of a new development tool. Existing development tools with UML modelling capabilities will suffice with only minor extensions. First of all the tool needs to be able to generate DAML-S, DAML+OIL and OWL from UML and it needs to be able to access UDDI registries and ontology repositories in an integrated fashion.

For ontology modelling there exists tools such as Protégé [PROTÉGÉ] which has a plug-in supporting OWL. Protégé lets you view a diagrammatic representation of the ontologies, which could be realized by UML diagrams, combining the worlds of ontology development and Web Service composition development. In order to accomplish this we need a better alignment of

ontologies and UML. The Object Management Group has issued a request for proposal for an ontology definition metamodel [OMGONTO], which might result in better support for ontology development using UML. Including such capabilities into a modelling tool will make it possible to follow the MOSIS approach to Web Service composition.

Since the discovery process is a sequential process with an unlimited number of Web Services that you want to discover, you will want to have the tool handling all the requests to UDDI registries and presenting you with the results. The reason for this is that the amount of information involved is too great to keep in separate places, you want all the information in one place. Imagine for instance that you are creating a composition that involves calls to ten Web Services, and all these can be fulfilled by a set of three alternative Web Services, this makes updates in the discovery process tedious if you have to fill in new information for each service you discover.

7.7.2 Platform independent repository

Only extending development tools facilitates a realization which is not optimal. Tool independent realizations will improve interoperability by acknowledging the distributed setting. As a consequence of this we envision a platform independent repository in which the metamodels and metadata of a Web Service composition exist. This repository will be used both in the development phase and after the composition has been deployed. The metamodel in the platform independent repository will be used to generate different views of the composition, such as an UML view or an ontology representation language view.

For projects under development the repository will serve as an integrating factor for the different aspects of composition development. Ontology tools can be used to develop the application ontology, which will then be accessible from UML tools used to develop the composition. Different tools can just access the model which will be in a form that is open and platform independent. Not relying on any proprietary formats increases the interoperability since it doesn't favor any tools, all tools can conform to the platform independent model. Figure 25 shows how different tools can work on the platform independent metamodel in the repository in a manner that ensures that all the metadata is updated and accessible by all tools in an integrated fashion.

After deployment, other compositions under development can use the information contained in the repository. When a Web Service is discovered, access to the application ontology will be through the repository, along with models that might be useful for the understanding of the semantics of the service. For instance, a model of the process might in some case be more useful than the DAML-S representation of the process. The platform independent metamodel will be used to provide different views of the metadata in the repository. Two important views in this context is the graphical UML view which is useful for developers, and an ontology view which is useful for machine interpretation of the semantics. Figure 25 shows how the repository will provide different representations of the metadata.

With respect to the semantical layer of a UDDI the repository will be complimentary. The DAML-S/UDDI matchmaker provides a registry which makes it possible to discover Web Services, and the repository contains the actual Web Service with models and views which may be useful for other

developers. Different views of the application ontology used will indeed provide developers with semantics useful for defining mappings.

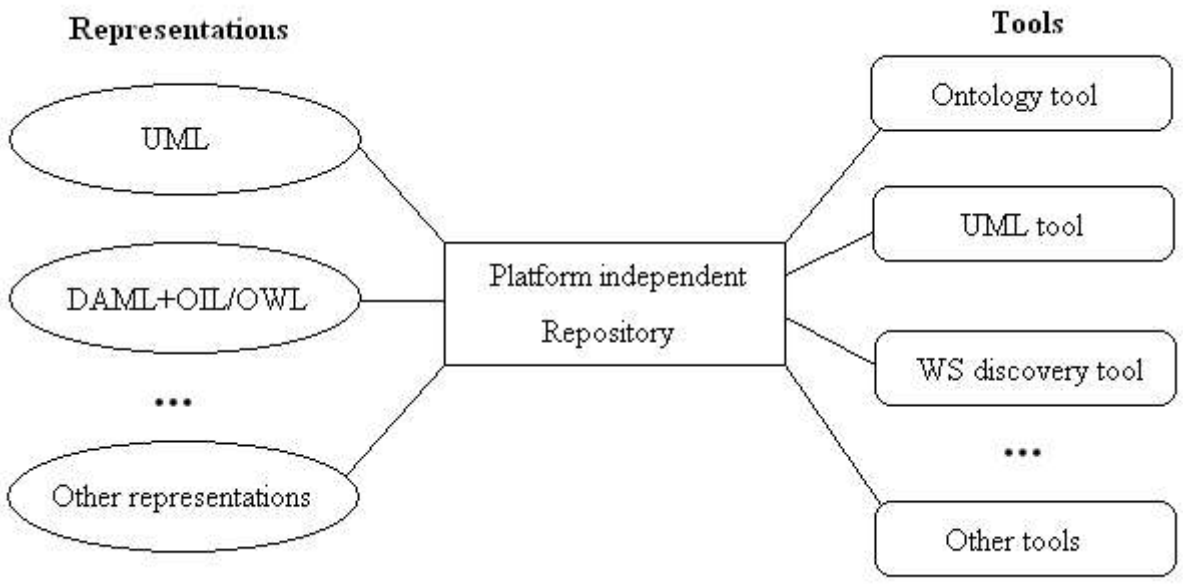

Figure 25 - Platform independent repository

Enabling technologies for the repository are OMGs Meta-Object Facility (MOF), which is an abstract language and a framework for specifying, constructing and managing metamodels, and QVT (Query/Views/Transformations) a transformation language that can express transformation rules between MOF compliant models.

7.8 Summary

This chapter presented MOSIS, our proposal for dealing with Web Service compositions. We explained how MOSIS deals with both the discovery and the composition phase with the use of a discovery model and a composition model. Central in MOSIS is also principles from MDA, such as platform independence and code generation, both helping developers unfamiliar with ontology representation languages and providing resistance to technology changes. We also improved the situation form the DAML-S approach where there are no guidelines for the use of ontologies by using a three level ontology architecture. Finally we included some thoughts on how MOSIS can be realized.

8. Using MOSIS

This chapter will guide us through MOSIS with help of the MODUSA case example presented in chapter 5. We'll see how the discovery model will look, and how the information contained in the information discovery model can be used to generate DAML-S profile descriptions to be used in the discovery of Web Services. After that we'll see how the composition model will look and how it can be used to generate INESC workflow XML in order to let a workflow engine produce code that executes and handles control and data flow.

8.1 The discovery model

In the MODUSA case example we identified four tasks that where suitable for being performed by a Web Service. Retrieving chemical information, weather information, a map of the area of concern and a calculation of the gas plume. Figure 26 shows the Web Service discovery UML model for the discovery process with the Web Service calls and information from the information discovery model identified.

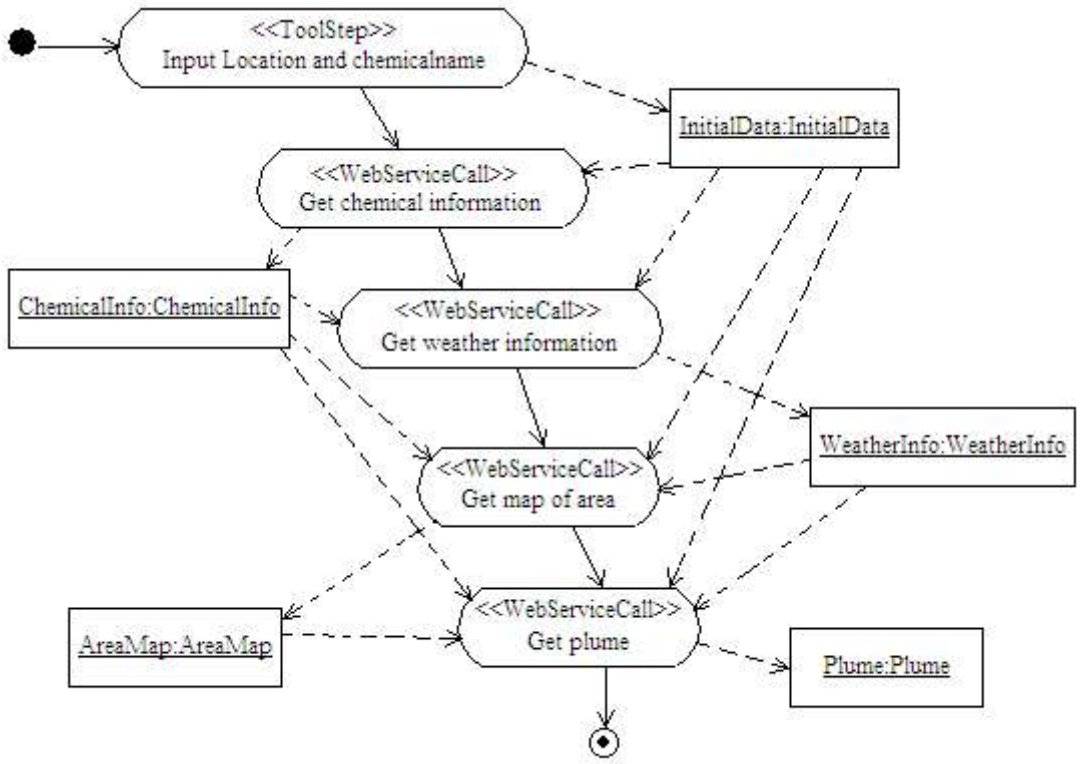

Figure 26 - The discovery model for the MODUSA case example

The first step is a Toolstep indicating that a human user is involved and provides the initial data, which is the location of the accident and the chemical involved. The next step is finding a Web Service that provides information about the chemical involved, and this service has available the location and the name of the chemical. The location is probably not relevant in this situation, but for the discovery phase, this information is available anyway just in case it might deem to be useful. The next step after this is getting weather information for the area of the accident. Here you still have the initial information available and in addition you have the chemical information available as well. Then you need Web Service providing a map of the area, which has available all the prior information in addition to weather information. The last step is calculating the gas plume, and here you have available all the information that is available from the whole process. The last step is the step that makes use of most of the information available.

8.1.1 Generating a DAML-S profile from the discovery model

Using the information from the Web Service information discovery model we want to generate the necessary DAML-S descriptions in order to discover the required services. In order to illustrate this we have chosen the weather service required in the MODUSA case example. Figure 27 shows the information from the information discovery model available for the weather service with links to elements/concepts in the application ontology developed for the discovery process describing the different attributes. The application ontology will exist in another diagram in the information discovery model, and will contain references to appropriate domain ontologies. The information is in the figure labeled with input and output and will be used as input and output properties in the DAML-S profile model. No preconditions or postconditions are defined for the weather service or any of the other services.

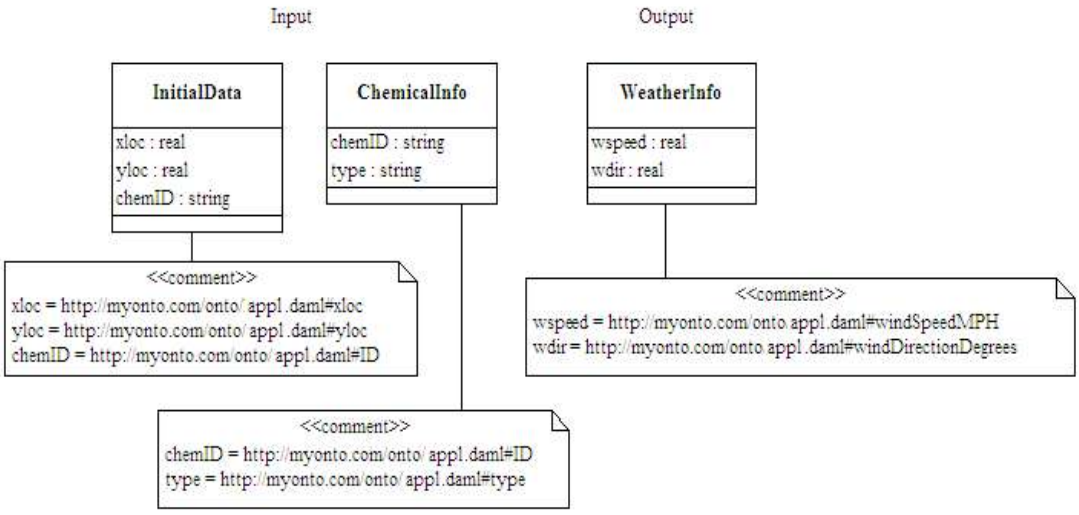

Figure 27- Part of the information discovery model showing available information and expected output for the weather service.

Below we have stated the input and output descriptions of a DAML-S profile model that would result from the generation of a description of the requested weather service in figure 26 with the descriptions of the information given in figure 27. The restrictedTo elements are in abbreviated form where &appl; refers to http://myonto.com/onto/appl.daml.

```
DAML-S profile model of the requested Weather service

<input>
    <profile:ParameterDescription rdf:ID=”Location_xloc”>
        <profile:parameterName> Latitude point </profile:parameterName>
        <profile:restrictedTo rdf:resource=”&appl;#xloc” />
    </profile:ParameterDescription>
</input>

<input>
    <profile:ParameterDescription rdf:ID=”Location_yloc”>
        <profile:parameterName> Longitude point </profile:parameterName>
        <profile:restrictedTo rdf:resource=”&appl;#yloc” />
    </profile:ParameterDescription>
</input>

<input>
    <profile:ParameterDescription rdf:ID=”ChemID”>
        <profile:parameterName> Chemical ID </profile:parameterName>
        <profile:restrictedTo rdf:resource=”&appl;#chemID” />
    </profile:ParameterDescription>
</input>

<output>
    <profile:ParameterDescription rdf:ID=”WindSpeed”>
        <profile:parameterName> WindSpeed </profile:parameterName>
        <profile:restrictedTo rdf:resource=”&appl;#wspeed” />
    </profile:ParameterDescription>
</output>

<output>
    <profile:ParameterDescription rdf:ID=”WindDirection”>
        <profile:parameterName> WindDirection </profile:parameterName>
        <profile:restrictedTo rdf:resource=”&appl;#wdir” />
    </profile:ParameterDescription>
</output>
```

The output parameters for the weather service are described by the application ontology which has references to a weather ontology, and we have included parts of the application ontology and parts of the weather domain ontology here. The application ontology represents the information in figure 27. Since there are no DAML-S elements describing precisely the reference to a domain we have used a non-existing element, DO_Reference, to illustrate the reference without defining it anywhere. The weather ontology is from the Agentcities web project at the University of Aberdeen [AC], it is not intended to be the complete and unique domain ontology for weather phenomena, but is included as an example of how a domain ontology can look.

Below follows parts of the application ontology defining the information in the WeatherInfo object in figure 27.

```
The information requested by a weather service in the application ontology

<xsd:simpleType name="positiveDouble">
    <!-- positiveDouble represents a positive decimal number -->
    <!-- meaning that the values must be >= 0 -->
    <xsd:restriction base="xsd:double">
        <xsd:minInclusive value="0" />
    </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="directionDegree">
    <!-- directionDegree represents a positive decimal number -->
    <!-- with the added restriction that values must be >= 0 and <= 360 -->
    <xsd:restriction base="xsd:double">
        <xsd:minInclusive value="0" />
        <xsd:maxInclusive value="360" />
    </xsd:restriction>
</xsd:simpleType>

<daml:Class rdf:ID="weatherInfo">
    <rdfs:comment>Superclass for the weatherInfo parameter</rdfs:comment>
</daml:Class>

<daml:Property rdf:ID="wspeed">
    <rdfs:label>Wind speed</rdfs:label>
    <rdf:type rdf:resource="daml:UniqueProperty" />
    <rdfs:domain rdf:resource="weatherInfo" />
    <rdfs:range rdf:resource="positiveDouble" />
    <refs:DO_reference="http://domonto.org/weather.daml#windspeed" />
</daml:Property>

<daml:Property rdf:ID="wdir">
    <rdfs:label>Wind direction</rdfs:label>
    <rdf:type rdf:resource="daml:UniqueProperty" />
    <rdfs:domain rdf:resource="weatherInfo" />
    <rdfs:range rdf:resource="directionDegree" />
    <refs:DO_reference="http://domonto.org/weather.daml#windDirection" />
</daml:Property>
```

Here we have defined that the wind speed must be a positive decimal number and that the wind direction must be a positive decimal number from 0 to 360. In addition we have defined that both wind speed and wind direction are unique properties, meaning that the wind will only have one speed and one direction. The restriction on the wind direction is too strict to find all Web Services that provide wind speed, but the discovery will mainly be based on the reference to the domain ontology, and the restriction in the application ontology can be used to select among the discovered Web Services. The references above to the domain ontology is to the weather domain ontology which we present parts of below.

```
Weather ontology describing wind from [AC]

<daml:Class rdf:ID="WindEvent">
    <rdfs:comment>Superclass for all events dealing with wind</rdfs:comment>
    <rdfs:label>Wind event</rdfs:label>
    <rdfs:subClassOf rdf:resource="#WeatherEvent" />
</daml:Class>

<daml:Property rdf:ID="windDirection">
    <rdfs:label>Wind direction</rdfs:label>
    <rdfs:domain rdf:resource="#WindEvent" />
</daml:Property>

<daml:Property rdf:ID="windSpeed">
    <rdfs:label>Wind speed</rdfs:label>
    <rdfs:domain rdf:resource="#WindEvent" />
</daml:Property>

<daml:Property rdf:ID="knotsSpeed">
    <rdfs:label>Speed of something, in knots</rdfs:label>
</daml:Property>

<daml:Property rdf:ID="mphSpeed">
    <rdfs:label>Speed of something, in MPH</rdfs:label>
</daml:Property>

<daml:Property rdf:ID="windSpeedMPH">
    <rdfs:subPropertyOf rdf:resource="#windSpeed" />
    <rdfs:subPropertyOf rdf:resource="#mphSpeed" />
</daml:Property>

<daml:Property rdf:ID="windSpeedKnots">
    <rdfs:subPropertyOf rdf:resource="#windSpeed" />
    <rdfs:subPropertyOf rdf:resource="#knotsSpeed" />
</daml:Property>

<daml:Property rdf:ID="windDirectionDegrees">
    <rdfs:comment>Direction of wind, in degrees</rdfs:comment>
    <rdfs:label>Wind direction (degrees)</rdfs:label>
    <rdfs:subPropertyOf rdf:resource="#windDirection" />
</daml:Property>

<daml:Property rdf:ID="windDirectionEnglish">
    <rdfs:comment>Direction of wind, in English</rdfs:comment>
    <rdfs:label>Wind direction (degrees)</rdfs:label>
    <rdfs:subPropertyOf rdf:resource="#windDirection" />
</daml:Property>
```

The domain ontology defines such things as wind speed and wind direction without data types and data models. Wind speed is defined for knots and miles per hour, and wind direction is defined in degrees and in plain English. In the application ontology the references were to the more specific properties, such as windSpeed instead of windSpeedMPH, in order not to put to strict restrictions resulting in potentially fewer services being discovered.

8.2 The composition model

The Web Service composition model includes the actual Web Services that will be used in the composition and is used to generate INESC workflow XML that a workflow engine can transform into code that executes the Web Services and handles the control and data flow.

Figure 28 shows the composition of the MODUSA case example with the Web Services we included in the client. The user supplies location and chemical name which is mapped into the chemical ID needed by the chemical information Web Service and into the three different location parameters needed by the weather service, the area map service and the plume service. The chemical information and the weather information also goes through a data mapper in order to fit the data expected by the plume service.

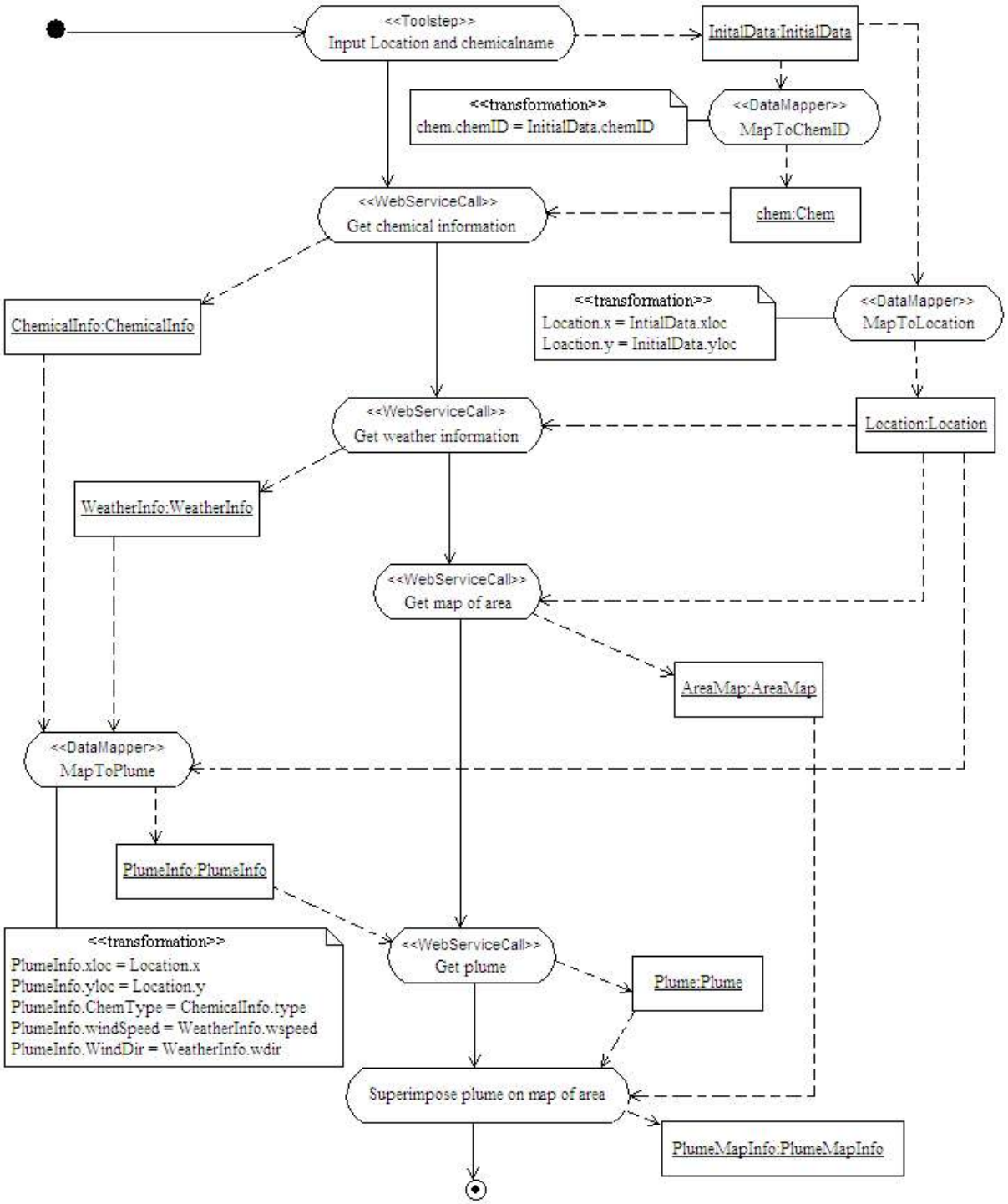

Figure 28 - The Composition model for the MODUSA case example

The model in figure 28 will be the basis for generating workflow XML, and for readers interested in the transformation we refer to [ACEGIS2].

Developing the mappings between the services will become easier since the discovery phase will have provided us with input and output that match semantically for serial composition. The information provided by the application ontologies of the services in a serial composition should provide us with enough to determine the transformation needed. For instance, if a weather service gives us wind direction in degrees blowing from a point, and another service needs wind direction in an 8-point compass (north, south, east, west, northeast, southeast, northwest, southwest) blowing to a point, this information should be captured in the application ontology in order to make it clear how to perform the transformation. In this case we are mapping to a more coarse domain so we will not lose information. The transformation will be, 337.5 to 360 degrees and 0 to 22.5 degrees is north, 22.5 to 67.5 degrees is northeast, 67.5 to 112.5 degrees is east, 112.5 to 157.5 degrees is southeast, 157.5 to 202.5 degrees is south, 202.5 to 247.5 degrees is southwest, 247.5 to 295.5 degrees is west and 295.5 to 337.5 degrees is northwest.

8.3 Summary

This chapter illustrated MOSIS using the MODUSA case example from chapter 5. We showed how the discovery model is used to generate a DAML-S profile model that can be used for Web Service discovery. In addition we showed how we can use the three layer ontology to describe information for discovery and to develop mappings in the composition phase. Finally we showed how the composition model defines the composition in a way that makes it possible to generate input to a workflow engine.

Part III

–

Conclusions and further work.

9. Conclusion

In this last chapter we'll sum up the implications of MOSIS by evaluating it in terms of the requirements we defined in chapter 6 and draw some conclusions from this evaluation. Furthermore, we'll identify some issues worth further investigation in section 9.2.

9.1 Evaluating MOSIS in terms of the requirements

The requirements we defined in chapter 6 provide a good way of evaluating MOSIS to see if the approach actually will improve the situation for Web Service compositions. As we concluded in chapter 6, one of the most inhibiting factors were the difficulty in understanding the descriptions of Web Services that are based on ontology representation languages. This situation we feel we have improved by introducing UML models that reflect these descriptions.

9.1.1 Evaluating MOSIS in terms of requirement 1

R1 – Change tolerance:	Web Service compositions that are set up to perform a specific task repeatedly will have to be tolerant to change in order to achieve interoperability.

By allowing for multiple services that can be set up to perform the same task we have a contingency plan in case one of the services is temporally unavailable. The `<<MultiServiceProviders>>` stereotype indicates that the task can be performed by one of a set of services, and the generated code from the workflow engine will implement this. Of course all eventualities cannot be accounted for, and in some cases all services for one task might be unavailable. In these cases the composition will break down, and one can only hope that this is not a frequent situation. To fully fulfill this requirement MOSIS will have to make use of automatic discovery for those cases where no service is available for a task.

9.1.2 Evaluating MOSIS in terms of requirement 2

R2 – Functionality matching:	Web Service discovery needs to be done by matching of functionality.

Since MOSIS makes use of the DAML-S approach to Web Service discovery, the discovery is done by matching required functionality with functionality of existing Web Services. By using domain ontologies as a basis for the discovery, MOSIS ensures that the discovery won't be to restricted, risking that potential Web Services will not be found. The actual success of the matching will be determined by the matchmaking software, either stand-alone or as an added semantic layer to UDDI.

9.1.3 Evaluating MOSIS in terms of requirement 3

R3 – Description types:	The means of representing a description of a required Web Service needs to find the right balancing point between human-understandable and machine-understandable.

This is the requirement we feel that MOSIS contributes most to, by modelling the discovery and the composition in UML that can be used to generate the necessary ontology representation language descriptions, both DAML-S profiles and application ontologies. Keeping both representations of the semantics, one in a human interpretable form and one in a machine interpretable form we get the best from both worlds. The machine interpretable form is necessary to make logical inferences to determine if two elements have the same meaning. The human interpretable form is necessary for the developers to focus on the important issues, and not only drown in details of an ontology

representation language. With an alignment between UML and ontology representation languages which provides a 1-1 mapping between the two the fulfillment of this requirement will be improved. Complete fulfillment of this requirement will involve making use of natural language to describe the requirements, thus involving the third group of users, normal users.

9.1.4 Evaluating MOSIS in terms of requirement 4

R4 – Operation order:	The order of execution among the operations in a Web Service needs to be defined.

As a result of being based on DAML-S MOSIS can interpret the order of the operations since it will be defined in the DAML-S descriptions. The DAML-S process model of the services that will be used in the composition can describe any specific ordering.

9.1.5 Evaluating MOSIS in terms of requirement 5

R5 – Service order:	The order of invocation among the Web Services in a composition needs to be defined.

The MOSIS Web Service composition model clearly states the order of the Web Services in the composition, and since this model will be used to generate workflow XML to be fed to a workflow engine, the actual order of invocation will be dealt with and maintained by the workflow engines output.

9.1.6 Evaluating MOSIS in terms of requirement 6

R6 – Semantic descriptions:	Web Services input and output must be described in a way that makes it possible to establish semantic similarity between them.

By following the three level ontology architecture we have a clear strategy of establishing semantic similarity. Domain ontologies will indicate if there is a semantic relation and application ontologies will indicate whether the additional restrictions makes it possible to define a transformation from output to input of services in a serial composition. Pending the specification of a semantic grounding the three level architecture is not complete and potential semantic mismatches may result from incomplete domain ontologies. In addition matching of elements from different domains will be difficult.

9.1.7 Evaluating MOSIS in terms of requirement 7

R7 – Semantic Web alignment:	Web Service composition approaches have to be in alignment with the Semantic Web vision.

In making use of DAML-S and DAML+OIL ontologies, MOSIS is in alignment with the "Semantic Web" vision. Upgrading to OWL and OWL-S, which are artifacts of the "Semantic Web" will involve writing a new generation tool that generates code in these languages instead of DAML-S and DAML+OIL.

9.1.8 Summary of the evaluation

Table 12 shows a summary of the evaluation of MOSIS. The potential for improvement from the evaluation of the existing approaches has been dealt with. The fulfillment of all the requirements have been improved even if all points aren't fully fulfilled.

	MOSIS
R1 – Change tolerance:	+/-
R2 – Functionality matching:	+
R3 – Description types:	+/-
R4 – Operation order:	+
R5 – Service order:	+
R6 – Semantic descriptions:	+/-
R7 – Semantic Web alignment:	+

Table 12 - Evaluation of MOSIS

Three of the requirements are only partially fulfilled by MOSIS, however by utilizing UML in the discovery model and the composition model we have weakened the barrier put up by ontology representation languages making use of ontologies easier for the average developer.

We have especially focused on the Web Service discovery issue, which is treated rather lightly by many involved with Web Services, and feel that our discovery model is a great step towards semantic discovery. The functionality matching provided by DAML-S is too cumbersome to become popular throughout the Web Service community without some measures of hiding the ugly details. Using UML in this phase lets the developers use a familiar technology that doesn't require new knowledge, in terms of ontology representation languages, that can be taken care of by appropriate tools.

The three level ontology architecture requires that application ontologies will be developed by the developers involved in the specific Web Service composition, and because of this letting developers use UML instead of ontology representation languages will improve the situation since UML is a more familiar technology for developers. In addition introducing the three level ontology we provide a strategy for the use of ontologies in Web Service composition. Only using ontologies developed in parallel with the Web Service creates a situation where matching may or may not be possible. With the three level ontology this situation will be amended.

9.2 Further work

The three requirements that MOSIS only partially fulfill represents the three areas where further work is required for MOSIS to become a complete solution for dealing with Web Service compositions. These areas are the three level ontology architecture, the alignment of UML and ontologies and the issue of automatic Web Service discovery.

The ontology issue is the main issue of concern for further work, and is a part of all three of the requirements that are not fully fulfilled. For the three level ontology architecture application ontologies and domain ontologies are constructs we are able to create as of today, but the third level, the grounding level, still requires more research to be constructed. If and how we can define the pattern or schemas that will represent more basic knowledge. Without the semantic grounding complete domain ontologies cannot be developed as they will be based on concepts from different generic ontologies, which will making matching rely on the ability to perform interontology matching. If domain ontologies reference concepts in the same or related semantic groundings, matchings will be made easier.

UML and ontologies are not in perfect alignment, there are ontology constructs that are difficult to represent in UML and vice versa. The work on defining specifications for a MOF2 metamodel and a UML2 profile that will support an alignment of UML and ontologies as a response to OMGs request for proposal in [OMGONTO], will probably improve the situation. If the specifications are successful UML modelling tools will be capable of developing ontologies. With an alignment of UML and ontologies the generation of ontology descriptions will be more complete, and transformations from a repository giving different views will be possible.

Until automatic Web Service discovery is mastered reacting to changing environments will rest on developers ability to find and set up alternative services for some task. In addition general purpose systems, such as problem solving agents, which require that services are discovered dynamically will not be possible until this issue is resolved. The lessons learned for discovery for specific purpose systems and the mastering of them, especially concerning ontologies, will probably be influential in resolving this issue.

10. Bibliography

[**AC**]: Agentcities at University of Aberdeen, *"Weather ontology"*, Agentcities project [online], 2002, URL: http://www.csd.abdn.ac.uk/research/AgentCities/WeatherAgent/weather-ont.daml, [quoted: 15.12.2003]

[**ACEGIS**]: ACE-GIS project team, *"Annex 1 - Descriptions of Work"*, ACE-GIS – Adaptable and Composable E-commerce and Geographic Information Services IST-2001-37724, 2002, URL: http://www.acegis.net

[**ACEGIS2**]: Roy Grønmo and David Skogan, *"Model-Driven Service Composition"*, ACE-GIS – Adaptable and Composable E-commerce and Geographic Information Services IST-2001-37724, 2003, URL: http://www.acegis.net

[**ACEGIS3**]: F. Probst, W. Kuhn and P. Maué, *"Proposal for extending W3C and OGC standards with semantic modelling capabilities - Formal Model Extension to Handle Evolving Semantics"*, ACE-GIS – Adaptable and Composable E-commerce and Geographic Information Services IST-2001-37724, 2003, URL: http://www.acegis.net

[**AHDIC**]: Editors of The American Heritage Dictionaries, "*The American Heritage Dictionary of the English Language*" [online], URL: http://www.bartleby.com, [quoted: 13.06.2003]

[**AL01**]: Alessio Lomuscio, *"Introduction - Intelligent Agent Issue"*, ACM Crossroads Student Magazine [online], 2001, URL: http://www.acm.org/crossroads/xrds5-4/intro54.html, [quoted: 10.01.2004]

[**AAAI02**]: American Association for Artificial Intelligence, *"Meaning Negotiation"*, MeaN-2002 - AAAI-02 Workshop on Meaning Negotiation [online], 2002, URL: http://boogie.cs.unitn.it/AAAI-02-MN/, [quoted: 26.04.2003]

[**BFM02**]: C. Bussler, D. Fensel and A. Maedche, *"A conceptual architecture for semantic web enabled Web Services"*, ACM SIGMOD Record, vol. 31 Issue 4, Dec. 2002, Pages: 24-29.

[**BPEL4WS**]: F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana, *"Specification: Business Process Execution Language for Web Services - Version 1.0"* [online], July 2002, URL: http://www-106.ibm.com/developerworks/library/ws-bpel1/, [quoted: 15.01.2004]

[**COMET**]: A. J. Berre, B. Elvesæter, J. Ø. Aagedal, A.Solberg, J. Oldevik and B. Nordmoen, *"COMET - Methodology Handbook with documentation of the COMET metamodels – Version 2.3"* [online], Feb. 2003, URL: http://www.ifi.uio.no/~mmo/generic/handbooks/COMET_v23.pdf, [quoted: 24.09.2003]

[**DAMLS**]: A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne and K. Sycara, *"DAML-S: Web Service Description for the Semantic Web"*, In The First International Semantic Web Conference (ISWC) Proceedings, Pages: 348-363, URL: http://www.daml.org/services/ISWC2002-DAMLS.pdf

[**DAMLS2**]: DAML-S Coalition, *"DAML Services"* [online], 2003, URL: http://www.daml.org/services/, [quoted: 11.11.2003]

[**F01**]: Dieter Fensel, *"Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce"*, Springer, 2001

[**FOLDOC**]: Denis Howe, *"The free online dictionary of computing"* [online], URL: http://www.foldoc.org, [quoted: 13.06.2003]

[**FSS03**]: K. Falkovych, M. Sabou and H. Stuckenschmidt, *"UML for the Semantic Web: Transformation-Based Approaches"*, In: B. Omelayenko and M. Klein – editors, Knowledge Transformation for the Semantic Web, IOS Press, 2003, Pages: 92-106, URL: http://www.cs.vu.nl/~heiner/public/KTSW.pdf

[**GYP02**]: Xiang Gao, Jian Yang and Mike. P. Papazoglou, *"The Capability Matching of Web Services"*, IEEE Fourth International Symposium on Multimedia Software Engineering (MSE'02), 2002, Pages: 56-63

[**IEEE90**]: Institute of Electrical and Electronics Engineers, *"IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries"*, New York, IEEE, 1990

[**ITS**]: The Intstitute for Telecommunication Sciences, *"Telecommunications: Glossary of Telecommunication Terms"* [online], 1996, URL: http://www.its.bldrdoc.gov/fs-1037/dir-019/_2838.htm, [quoted: 13.06.2003]

[**K94**]: Espen Frimann Koren, *"Semantic proximity in Object-oriented Data Models"*, Master thesis, Department of Informatics, University of Oslo, 1994

[**K96**]: Steinar A. Kindingstad, *"ODL-M, A Mapping Language for Schema Integration in Object-Oriented Multidatabase Systems"*, Master thesis, Department of Informatics, University of Oslo, 1996

[**L02**]: George F. Luger, *"Artificial Intelligence - Structures and strategies for complex problem solving"*, 4th edition, Pearson Addison Wesley, 2002

[**LHL01**]: Tim Berners-Lee, James Hendler and Ora Lassila, *"The Semantic Web"*, In Scientific American, May 2001, Pages: 34-43

[**MAS**]: Roberto A. Flores-Mendez, *"Towards a Standardization of Multi-Agent System Frameworks"*, ACM Crossroads Student Magazine [online], 2001, URL: http://www.acm.org/crossroads/xrds5-4/multiagent.html, [quoted: 10.01.2004]

[**MBE03**]: B. Medjahed, A. Bouguettaya and A. Elmagarmid, *"Composing Web services on the Semantic Web"*, In the VLDB Journal — The International Journal on Very Large Data Bases, vol. 12 Issue 4, Nov. 2003, Pages: 333-351

[**MD02**]: Michael Denny, *"Ontology Building: A Survey of Editing Tools"*, O'Reilly XML.com [online], Nov. 2002, URL: http://www.xml.com/pub/a/2002/11/06/ontologies.html, [quoted: 14.04.2003]

[**MDAF03**]: David S. Frankel, *"Model Driven Architecture – Applying MDA to Enterprise Computing"*, Wiley Publishing Inc., 2003

[**MM03**]: Daniel J. Mandell and Sheila A. McIlraith, *"Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation"*, 2nd International Semantic Web Conference, 2003,
URL: http://www.ksl.stanford.edu/people/sam/iswc2003sam-djm-FINAL.pdf,
[quoted: 10.01.2004]

[**O00**]: Uche Ogbuji, *"An introduction to RDF - Exploring the standard for Web-based metadata"* [online], 2000, URL: http://www-106.ibm.com/developerworks/library/w-rdf/,
[quoted: 19.11.2003]

[**OMGMDA**]: Object Management Group, *"MDA"* [online], 2002,
URL: http://www.omg.org/mda/, [quoted: 13.05.2003]

[**OMGONTO**]: Object Management Group, *"Ontology Definition Metamodel - Request For Proposal"* [online], 2003, URL: http://www.omg.org/docs/ad/03-03-40.pdf,
[quoted: 15.01.2004]

[**P03**]: Chris Peltz, *"Web Service Orchestration - a review of emerging technologies, tools, and standards"*, Hewlett-Packard Techincal white papers [online], 2003,
URL: http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf,
[quoted: 17.10.2004]

[**PKI00**]: Steven Lloyd, Derek Brink, Andrew Nash, Gordon Buhle and Nada Kapidzic Cicovic, *"PKI Interoperability Framework"* [online], 2000,
URL: http://www.pkiforum.org/pdfs/PKIInteroperabilityFramework.pdf, [quoted: 13.04.2003]

[**PKPS02**]: M. Paolucci, T. Kawamura, T. Payne and K. Sycara, *"Importing the Semantic Web in UDDI"*, In Web Services, E-Business and Semantic Web Workshop, 2002

[**PROTÉGÉ**]: Stanford University, *"The Protégé Ontology Editor and Knowledge Acquisition System"* [program], 2003, URL: http://protege.stanford.edu/, [quoted: 17.12.2004]

[**RS97**]: Mary Tork Roth and Peter Schwarz, *"Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources"*, In Proceeding of the 23th VLDB Conference, 1997, Pages 266-275

[**SK92**]: Amit Sheth and Vipul Kashyap, *"So Far (Schematically) yet So Near (Semantically)"*, In Proceedings, IFIP Wg 2.6 Conference on Semantics of Interoperable Database Systems (Data Semantics 5), 1993, Pages 283-312

[**SL03**]: Declan O'Sullivan and David Lewis, *"Semantically Driven Service Interoperability for Pervasive Computing"*, In Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access, 2003, Pages 17-24

[**SL90**]: Amit P. Sheth and James A. Larson, *"Federated database systems for managing distributed, heterogeneous, and autonomous databases"*, ACM Computing Surveys (CSUR), vol. 22 Issue 3, Sep. 1990

[**SRS**]: Marta Sabou, Debbie Richards and Sander van Splunter, *"An experience report on using DAML-S"*, In The Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW), 2003,
URL: http://www.iids.org/publications/essw03.pdf

[**UDDI**]: The UDDI consortium, *"UDDI Technical White Paper"* [online], 2000, URL: http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf, [quoted: 12.09.2003]

[**W3COWL**]: World Wide Web Consortium (W3C), *"OWL Web Ontology Language Overview"* [online], URL: http://www.w3.org/TR/owl-features/, 2003, [quoted: 17.09.2003]

[**W3CRDF**]: World Wide Web Consortium, *"Resource Description Framework (RDF)"* [online], URL: http://www.w3.org/RDF/, 2003, [quoted: 17.09.2003]

[**W3CSOAP**]: World Wide Web Consortium, *"SOAP Version 1.2 Part 0: Primer"* [online], URL: http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, 2003, [quoted: 18.08.2003]

[**W3CSW**]: World Wide Web Consortium, *"Semantic Web"* [online], 2001 URL: http://www.w3.org/2001/sw/, [quoted: 21.10.2003]

[**W3CWSA**]: World Wide Web Consortium Web Services Architecture group, *"Web Services Architecture"* [online], 2003, URL: http://www.w3.org/TR/2003/WD-ws-arch-20030808/, [quoted: 18.08.2003]

[**W3CWSDL**]: World Wide Web Consortium, *"Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language"* [online], 2003, URL: http://www.w3.org/TR/wsdl20/, [quoted: 18.08.2003]

[**W3CXML**]: World Wide Web Consortium, *"Extensible Markup Language (XML)"* [online], 2003, URL: http://www.w3.org/XML/, [quoted: 17.09.2003]

[**WDL**]: Lee A. Benzinger, *"Applying the WDL composition theory to analyze security architectures"*, Composition Software Architecture Workshop, Jan. 1998, URL: http://www.objs.com/workshops/ws9801/papers/paper036.ps

[**WM01**]: William E. Moen, *"Mapping the Interoperability Landscape for Networked Information Retrieval"*, In Proceedings of the first ACM/IEEE-CS joint conference on Digital libraries, 2001, Pages: 50-51

[**WORDNET**]: Cognitive Science Laboratory at Princeton University, *"WordNet 1.7.1"* [online], 2003, URL: http://www.cogsci.princeton.edu/cgi-bin/webwn, [quoted: 13.06.2003]

[**WSI03**]: Web Services Interoperability Organization, *"Basic Profile Version 1.0a"* [online], 2003, URL: http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.html, [quoted: 19.08.2003]

[**WVV01**]: H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt,G. Schuster, H. Neumann and S. Hübner, *"Ontology-Based Integration of Information —A Survey of Existing Approaches"*, IJCAI-01 Workshop: Ontologies and Information Sharing, 2001, Pages: 108-117